uc3m

Ingeniería en Informática

2015-2017

Proyecto Fin de Carrera

“Desarrollo e implementación de un modelo de peatón para el simulador Webots”

Mariano Navarro Corrales

Tutor

Agapito Ismael Ledezma Espino

Palabras clave: Robotics, biped locomotion, genetic programming, differential equations

Descripción: Se trata de un desarrollo de un modelo de peatón para un entorno de conducción urbana en el simulador comercial Webots. El desarrollo de este controlador se realiza mediante la optimización de un modelo bípedo de locomoción basado en ecuaciones diferenciales mediante programación genética. También se han utilizado modelos ad-hoc.

Agradecimientos

Las condiciones en las que cada estudiante desarrolla su proyecto fin de carrera son tan diferentes y variadas que forman una sola unidad con el mismo proyecto, y con su autor. Por este motivo, me gustaría dar las gracias a aquellas personas que han estado a mi alrededor durante todo este tiempo y que han puesto su granito de arena para que no me faltase motivación hasta el final del proyecto.

En primer lugar, me gustaría agradecer a Mari la comprensión y el tiempo que ha dedicado a escuchar mis comentarios. Soy consciente de que en determinados momentos han podido resultar bastante incomprensibles y han requerido un esfuerzo notable para entender una parte del mensaje que pretendía transmitir.

En segundo lugar no puede faltar el agradecimiento a mis padres, a mi hermana y a mi familia, los cuales han hecho un notable esfuerzo apoyándome cuándo más lo necesitaba a su manera y dentro de sus posibilidades. Aprecio enormemente el esfuerzo que han realizado no sólo a lo largo del periodo de desarrollo del presente proyecto, si no hasta ahora tanto cuando comprendían mis avances como cuando estos se quedaban fuera de su comprensión pese a mis esfuerzos.

En tercer lugar, me gustaría reconocer y agradecer los comentarios tanto de interés como de ánimo expresados por mis amigos, en los que me mostraban su apoyo. En ambos casos, dichos comentarios estimulaban el ejercicio de desarrollo que he realizado en este proyecto fin de carrera. En especial, me gustaría agradecerles a Alberto, Ana, Elodie, Luisma y Carlos sus comentarios que me ofrecieron una segunda visión muy valiosa.

En cuarto lugar, también quiero agradecer a todos mis profesores la oportunidad y el tiempo que me brindaron para formarme como persona e ingeniero, sin cuya ayuda y conocimientos no habría recorrido este camino hasta presentar dicho proyecto. En especial,

me gustaría agradecer el soporte y los comentarios que me ha brindado mi tutor, Agapito, acerca del trabajo realizado y la redacción de la memoria. Gracias a su apoyo he concluido este proyecto.

En último lugar, no puedo cerrar esta sección sin agradecer de modo importante aquellas voces críticas, que pese a las dificultades y no siendo amables en el mensaje, siempre me dieron un punto de vista diferente al mío. Dichas críticas, que me aportaron otras referencias y riqueza, también me resultaron muy útiles.

Índice general

Índice de figuras

Capítulo 1

Introducción

Durante el desarrollo de este proyecto fin de carrera se observa el progreso realizado en la investigación y desarrollo de sistemas inteligentes adaptativos de asistencia la conducción y vehículos autónomos. Hasta hace poco tiempo el control de los vehículos lo realizaba completamente el conductor. Estas investigaciones están dando lugar a vehículos con capacidad para asumir más funciones propias de la conducción. A día de hoy algunos de estos tipos de vehículos ya circulan por nuestras calles y carreteras.

Los sistemas, que soportan estas funciones, permiten al vehículo detectar situaciones de peligro y avisar al conductor de éstas, o incluso, tomar decisiones apropiadas al entorno que rodea a éste. Sin embargo, las capacidades de éstos disponen aún de cierto grado de error y, por lo tanto, pueden necesitar un proceso de depuración y optimización.

En el desarrollo de nuevos sistemas y en el proceso de mejora de los ya existentes se utilizan peatones para poder probarlos. Para no poner en riesgo la vida de éstos se utilizan simuladores hasta que se considera que el sistema alcanza un nivel de madurez suficiente. En este proyecto se busca incorporar un modelo de peatón en un simulador de robótica comercial, concretamente, Webots.

Para incluir en la simulación un modelo de peatón, que se mueva de acuerdo a las leyes físicas, se ha utilizado un robot bípedo con proporciones y aspecto humanos. El control de su movimiento se ha realizado mediante diferentes programas, que se han creado empleando una técnica de aprendizaje automático, concretamente, la programación genética. La construcción de estos programas requiere de un ajuste complejo de las acciones a ejecutar y se utiliza esta técnica para encontrar el ajuste óptimo. La aplicación de esta

técnica se ha realizado mediante el uso, la modificación y la evolución de una herramienta de programación genética ya existente[35].

El principal objetivo de este proyecto consiste en desarrollar un controlador que permita desplazar un peatón virtual con aspecto, masa y proporciones similares a un ser humano desde un punto A a un punto B. No obstante, existen otros objetivos adicionales que se buscan con este proyecto. Entre esos objetivos adicionales se encuentra que el peatón debe recorrer esa distancia por el camino más corto. Además, se busca determinar el impacto del uso de sensores en el rendimiento ofrecido por el controlador y en el modo de caminar del peatón.

A continuación se lista un resumen de los objetivos:

- Adquirir el conocimiento necesario para conocer el problema y las diferentes aproximaciones existentes que pueden aplicarse para resolverlo.
- Crear un programa que desplace un peatón de un punto A a un punto B por el camino más recto, caminando de forma estable. Se considera alcanzado el objetivo, si el controlador resultante permite caminar al peatón durante más de un minuto.
- Crear un peatón virtual que tenga un aspecto similar a un peatón real.
- Adaptar una aplicación de programación genética existente a otro dominio dado, y reconocer las modificaciones necesarias para su extensión.
- Mejorar el desarrollo de la aplicación de programación genética para dotarla de mayor estabilidad, mediante la implementación de pruebas automáticas.
- Realizar los experimentos necesarios para estudiar el impacto entre el uso de sensores y no uso de éstos sobre las capacidades del controlador y su forma de hacer caminar al peatón.

No obstante, los siguientes objetivos quedan fuera del alcance del presente proyecto:

- Crear un peatón que pueda esquivar obstáculos.
- Crear un peatón que realice un seguimiento de la ruta indicada.
- Crear un peatón que pueda subir o bajar pendientes.

- Crear un peatón con capacidad para caminar a través de terrenos no lisos.

Una vez presentado el proyecto, se procede a detallar la estructura de la memoria. Ésta se encuentra dividida en los siguientes capítulos.

En el capítulo 2 se explica el estado del arte de las diferentes disciplinas que abarca el presente proyecto fin de carrera y se presentan algunos conceptos básicos para su comprensión.

El capítulo 3 muestra el desarrollo realizado para adaptar el algoritmo de programación genética existente a los nuevos requisitos, tanto de funcionalidad como de estabilidad. También presenta los resultados alcanzados en materia de verificación y estabilidad.

En el capítulo 4 se detallan los modelos bípedos de locomoción que se han probado, así como los resultados obtenidos mediante su uso. En este capítulo se muestran los resultados obtenidos en el proceso evolutivo con un modelo *adhoc*, así como aquellos obtenidos mediante un modelo basado en ecuaciones diferenciales.

El capítulo 5 lista las conclusiones a las que se ha llegado, una vez realizado el trabajo y analizados los resultados tanto del capítulo 3 como del 4.

En el capítulo 6 se presentan las posibilidades de continuación de la investigación y las líneas futuras que podrían seguirse, si se desease continuar con su desarrollo en el futuro.

En último lugar, se presentan los anexos que se han considerado de interés para el lector, de forma que pueda ahondar en estos aspectos.

Capítulo 2

Estado del arte

En este capítulo se describen las diferentes temáticas y técnicas que se encuentran relacionadas con este proyecto, para proporcionar al lector una visión de conjunto que le permita situar correctamente el desarrollo realizado y los objetivos propuestos.

Dentro de las temáticas, se pueden encontrar en primer lugar los sistemas avanzados de asistencia a la conducción (ADAS, por sus siglas en Inglés). En segundo lugar, se muestran los coches autónomos y los diferentes simuladores que existen en la actualidad, clasificados atendiendo a sus características.

En la parte relativa a las técnicas teóricas se hallan aquellas que se han empleado para el desarrollo del proyecto. En esta parte, se explican los modelos bípedos de locomoción, así como la programación genética. En ambos casos, se facilita una descripción con mayor detalle de ambas técnicas en su sección correspondiente.

2.1. Sistemas avanzados de asistencia a la conducción

A diario se producen millones de desplazamientos en carretera y vías urbanas por todo el mundo. En un porcentaje reducido de éstos se producen accidentes, en algunos casos con víctimas. Sin embargo, el número de víctimas en accidentes de tráfico resulta muy elevado. De esto se desprende que se necesitan vías urbanas e interurbanas que sean más seguras[1].

[1] No pretendo expresar que dependa sólo de las carreteras/calles, si no la conjunción de carretera y vehículos que circulan por ella.

Como respuesta a este importante problema existen los sistemas avanzados de asistencia a la conducción (*ADAS*, por sus siglas en Inglés). Dichos sistemas ofrecen una solución parcial para mejorar la seguridad en las carreteras. Esta se encuentra enfocada en varios objetivos.

En primer lugar, se busca facilitar información útil al conductor tan pronto esté disponible, permitiéndole tomar mejores decisiones. Este es el objetivo, por ejemplo, de aquellos *ADAS* que detectan otros vehículos fuera del ángulo de visión del conductor.

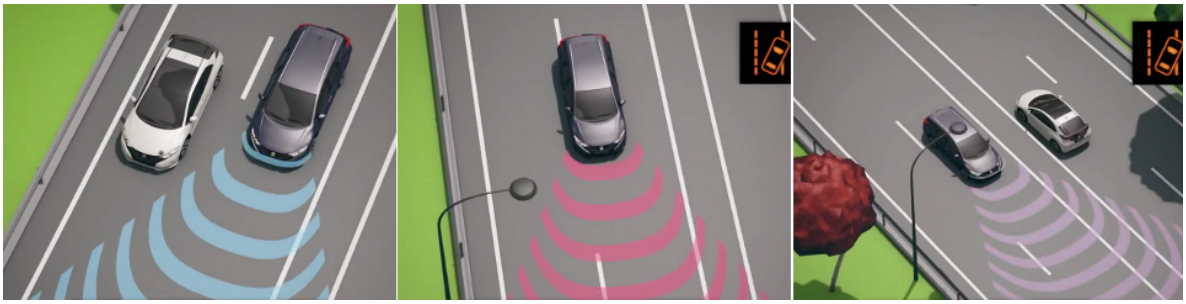
En segundo lugar, se persigue llamar la atención del conductor sobre eventos que pueden pasar desapercibidos, para evitar una posible situación de peligro. En este caso, se incluyen en esta categoría aquellos *ADAS* que permiten detectar un cambio de carril accidental, por ejemplo.

En último lugar, los *ADAS* tienen como misión intervenir en caso de emergencia, tomando el control una vez el conductor no muestra intención o no reacciona a tiempo de hacerlo. Es el caso de los *ADAS* que impiden una colisión, cuando un conductor circula por una vía urbana y no detecta a tiempo que el vehículo precedente frena de imprevisto.

En los dos primeros casos, los sistemas *ADAS* facilitan la información mediante alertas visuales y sonoras. En ningún caso éstos buscan sustituir al conductor, si no facilitar su conducción. A continuación se presentan diferentes tipos de *ADAS* y se describen brevemente cada uno de ellos.

Alerta por cambio involuntario de carril

Supóngase que un conductor circula por cualquier vía distinta a una calle de dirección única con un sólo carril. En esta situación un cambio de carril no deseado puede tener consecuencias, cuya gravedad dependerá del tráfico existente y su sentido.



HONDA / [1]

a) b) c)

Figura 2.1: Alerta de cambio de carril involuntario

a) Vehículo circulando por el centro del carril. Sistema detecta situación normal.
b) Vehículo pisando el carril derecho. Sistema detecta situación anómala.
c) Vehículo volviendo a circular por el centro del carril. Sistema vuelve a detectar situación normal.

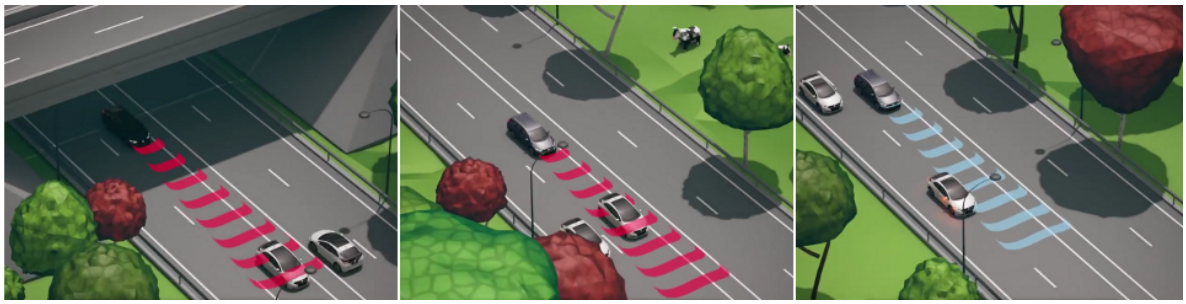
En vehículos no equipados con un sistema de éste tipo la capacidad de mantener el vehículo entre las líneas del carril depende exclusivamente de la pericia del conductor. No obstante, las distracciones pueden llevar a un conductor a salirse involuntariamente del carril y tener un accidente.

En vehículos equipados con este sistema (como muestra la Figura 2.1), éste permite alertar al conductor cuando se produce un cambio de carril no voluntario. Su funcionamiento se basa en la detección de las líneas de la carretera, y de otros vehículos que se encuentren en la vía. Además, detecta la ausencia de indicadores de aceleración del vehículo, y el estado de los indicadores de dirección del vehículo, para determinar que el cambio de carril no es voluntario.

La Figura 2.1.a muestra dos vehículos circulando en paralelo correctamente. En la Figura 2.1.b se observa como el vehículo en el carril de la izquierda acaba de dejar atrás al vehículo en el carril derecho, cuando pisa involuntariamente la linea de separación de los carriles. Éste *ADAS* responde activando una alerta que permite al conductor reorientar el vehículo dentro del carril. La Figura 2.1.c muestra al vehículo dejando de pisar la linea de cambio de carril.

Alerta por colisión frontal

Este sistema avisa al conductor cuando el radar delantero detecta una reducción súbita de la distancia al vehículo precedente. El funcionamiento de dicho sistema se produce por fases. En una primera fase, dicho sistema (véase la Figura 2.2) alerta de forma acústica y visual al conductor de la situación.



HONDA / [1]

a) b) c)

Figura 2.2: Alerta por colisión frontal

a) Vehículo circulando por autovía. Sistema detecta vehículo precedente a media distancia.
b) Vehículo se acerca demasiado. Sistema detecta inacción por parte del conductor. El sistema aplica frenado de emergencia.
c) Vehículo precedente abandona el mismo carril. Sistema libera frenado de emergencia.

En una segunda fase, que se produce muy pocos segundos después, el sistema aplica un frenado inicial para reducir la velocidad de aproximación del vehículo.

La Figura 2.2.a muestra a dos vehículos circulando por el mismo carril, y el primero circula con menor velocidad que el segundo en el sentido de la marcha. Al detectar que se está acercando al vehículo precedente, este *ADAS* lleva a cabo avisos acústicos y, eventualmente, realiza una reducción de la marcha (Figura 2.2.b). Cuando el vehículo precedente completa su adelantamiento y deja libre el carril, este *ADAS* permite retomar la marcha a la velocidad original (Figura 2.2.c).

Reconocimiento de señales de tráfico

La conducción de un vehículo es un proceso humano que requiere de toda la atención. En algunos momentos, incluso toda la atención del conductor puede no ser suficiente, y a éste se le pueden pasar señales de tráfico. El sistema descrito detecta la señal encontrada y muestra la señal detectada en el panel de mandos del vehículo.

Soporte para uso de luces de largo alcance

La conducción nocturna requiere, cuando se transita por vías poco iluminadas o desconocidas, del uso de luces de largo alcance, también conocidas como *largas*. Su uso indebido no sólo puede molestar a otros conductores, si no que puede ser la fuente de accidentes al provocar deslumbramientos e incluso motivo de sanción.

Para facilitar que los conductores realicen un uso responsable de las mismas, este sistema monitoriza mediante una cámara constantemente la escena. De esta forma, este sistema puede detectar vehículos en la misma dirección, así como en dirección contraria y cambiar las luces de forma automática para evitar este efecto indeseado. En la Figura 2.3 se puede ver una ilustración del funcionamiento del sistema.

HONDA / [1]

a) b) c)

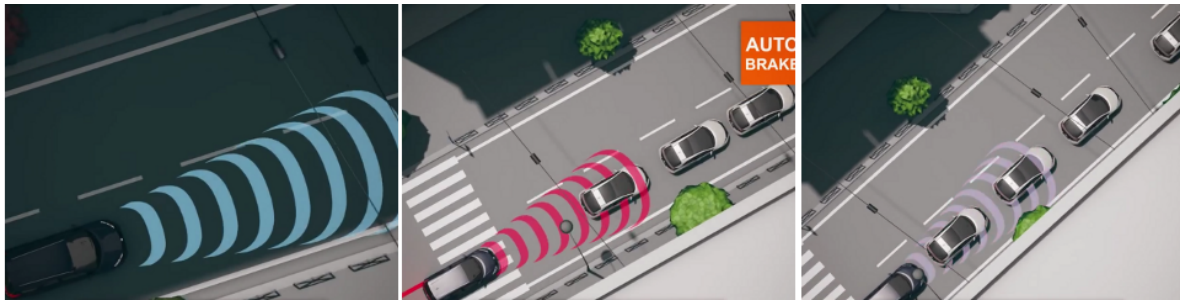
Figura 2.3: Soporte para uso de luces de largo alcance

a) Vehículo circulando por autovía. Sistema detecta vehículo precedente a media distancia.
b) Vehículo se encuentra vehículo circulando en sentido contrario. Sistema cambia el tipo de luces a luces de cruce.
c) Vehículo termina de cruzarse con vehículo contrario. Sistema vuelve a encender la iluminación de largo alcance.

En la Figura2.3.a se observa a un vehículo circulando por un tramo sin iluminar en una carretera de varios carriles en cada sentido. Cuando éste se encuentra con un vehículo en sentido contrario, el sistema detecta a dicho vehículo y cambia automáticamente a luces de cruce (Figura 2.3.b). Si el vehículo no dispone de este tipo de *ADAS*, el conductor debe cambiarlas manualmente. En la Figura2.3.c se ve como al completar el cruce con el vehículo que circula en sentido contrario, este *ADAS* vuelve a activar las luces de largo alcance.

Frenado autónomo de emergencia

El sistema tratado en esta sección tiene como finalidad detener el vehículo de forma autónoma ante un obstáculo. Supóngase que un conductor circula por una vía urbana con circulación (Figura 2.4.a). Si se produjese una reducción de la marcha, o incluso una detención, y el conductor no apreciase el cambio en la situación (Figura 2.4.b), un *ADAS* de este tipo puede llevar a cabo una reducción de la marcha o detener el vehículo (Figura 2.4.c).



HONDA / [1]

a) b) c)

Figura 2.4: Frenado autónomo de emergencia

a) Vehículo circulando por vía urbana.
b) Vehículo se encuentra vehículo circulando en mismo sentido, pero detenidos. Sistema detecta posible colisión frontal si se mantiene velocidad. Sistema acciona el freno ante la inacción del conductor.
c) Vehículo termina de realizar el frenado.

La utilidad de este *ADAS* se encuentra principalmente en la circulación en entornos urbanos, situaciones de retención o alta densidad de circulación.

Control de velocidad adaptativo

Este sistema permite al conductor especificar al coche una velocidad de crucero, y que ésta se adapte a las condiciones del tráfico. Por ejemplo, un conductor conduce por la izquierda con velocidad de crucero a 120 km/h y sale del carril de la derecha un vehículo con intención de adelantar. Si el primero de los vehículos tuviese un control de velocidad no adaptativo, éste tendría que frenar el vehículo (y desconectar el control de velocidad) para evitar acercarse demasiado o incluso impactar con el vehículo que le precede.Este sistema, no obstante, le permite al conductor no tener que tocar el freno. Es capaz de reducir la marcha para mantener la distancia de seguridad con el vehículo precedente. En cuanto dicho vehículo vuelve a la derecha, restaura la velocidad de crucero fijada con anterioridad.

Sistema de detección de tráfico perpendicular

Imagínese que un vehículo se encuentra estacionado en batería y que la parte delantera de éste da a la acera. Cuando el conductor comience la maniobra de salida, éste no dispone de referencias visuales para comprobar si existe tráfico circulando por la vía. Comúnmente, la forma de detectar si hay tráfico o no, se basa en el ruido que producen los vehículos con motor de combustión. Sin embargo, este enfoque no puede aplicarse con bicicletas. El sonido que producen éstas es insuficiente para que puedan ser detectados por los otros conductores. Dicho sistema permite detectar si hay tráfico circulando por la parte posterior del vehículo y avisa al conductor, incluso cuando el tráfico se encuentra compuesto por vehículos eléctricos.

HONDA / [1]

a) b) c)

Figura 2.5: Sistema de detección de tráfico perpendicular

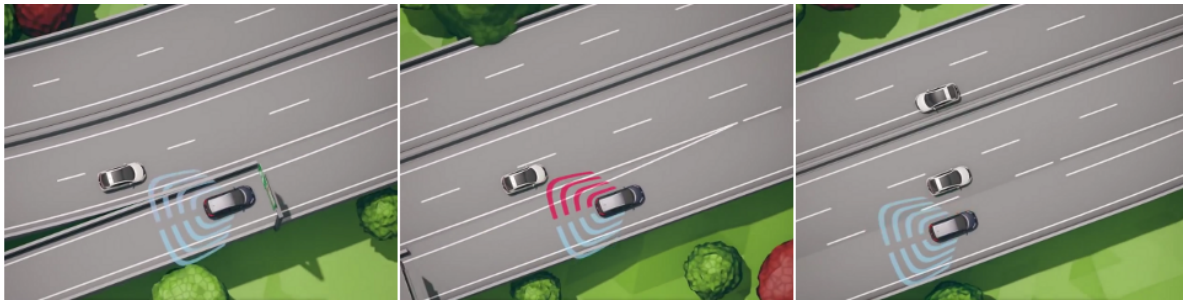
a) Vehículo circulando por vía urbana. Da marcha atrás para acceder a la vía.
b) Sistema detecta vehículo circulando por detrás en dirección perpendicular. Sistema alerta al conductor del evento.
c) Vehículo continúa realizando la maniobra de incorporación, una vez no se encuentra ningún vehículo en su trayectoria.

La Figura 2.5.a muestra a un vehículo saliendo marcha atrás de un estacionamiento. Al iniciar la marcha, el sistema no detecta vehículos en las proximidades. En el momento que se rebasa el muro que da acceso a la calle, éste *ADAS* detecta un vehículo (Figura 2.5.b) y alerta al conductor. Al terminar de pasar por detrás del vehículo, el *ADAS* detecta la nueva situación y comunica al conductor que puede proseguir con la maniobra (Figura 2.5.c).

El radar que incorpora el sistema detecta la presencia de vehículos aproximándose en el plano perpendicular y puede advertir al conductor de la circunstancia.

Asistente de punto ciego

Cuando se realiza un cambio de carril, los espejos retrovisores permiten controlar que la maniobra se realiza de forma segura. No obstante, hay un área a ambos lados del vehículo que los espejos no permiten ver (Figura 2.6). A esta zona se le denomina *punto ciego*.



HONDA / [1]

a) b) c)

Figura 2.6: Asistente de punto ciego

a) Vehículo circulando por vía interurbana. Realiza maniobra de incorporación a otra vía.
b) Sistema detecta vehículo circulando por detrás en el área de baja visibilidad junto al vehículo, o ángulo muerto. Sistema alerta al conductor del evento.
c) Sistema cesa la alerta al conductor, una vez el otro vehículo ha salido de la zona de baja visibilidad.

Para evitar que el conductor pueda ser sorprendido al realizar un cambio de carril y que otro vehículo pueda encontrarse en esta zona, este sistema emplea radares en ambas zonas para detectar dicha situación y advertirnos del peligro existente. Al detectarlo, el sistema avisa al conductor mediante un aviso acústico y otro visual.

Asistente de control de atención

Este sistema resulta de particular utilidad en los desplazamientos de larga duración. La situación de estudio parte de que un conductor se encuentra conduciendo durante varias horas en un viaje. Como resultado de la fatiga al volante, comienza a sufrir efectos no deseados, tales como una desviación en la trayectoria, cambios de carril no deseados y pérdida de atención al volante.

En condiciones en las que el asistente de control de atención no se encuentra disponible, el conductor es el único responsable de determinar el momento idóneo de realizar un alto en el camino.

Si se encuentra disponible este asistente, el coche puede comparar el estilo de conducción del conductor con y sin fatiga y determinar en que momento empiezan a ocurrir los efectos de la fatiga al volante. Este sistema, además, puede indicar al conductor que debe

realizar una pausa para reducir la fatiga y reducir la probabilidad de que estos efectos se produzcan.

Alerta de colisión con peatón

Imagínese un vehículo circulando por una zona urbana con vehículos aparcados a ambos lados de la vía a punto de llegar a un paso de peatones. En el paso de peatones no se observa ningún peatón con intención de cruzar. Mientras el conductor aparta la vista de la vía durante un instante, aparece un peatón de forma repentina.

Si no se dispone de un sistema de asistencia que detecte los peatones, el conductor tendría que dar un frenazo para evitar un accidente, puesto que no sería consciente de la presencia del peatón hasta que volviese a mirar a la carretera.

Sin embargo, un sistema de este tipo puede advertir de la situación tan pronto sucede. Esta diferencia de tiempo entre el caso anterior y éste puede permitir al conductor realizar una frenada a tiempo en lugar de una de emergencia.

2.2. Coches autónomos

Los vehículos autónomos son un tipo de automóvil capaz de asumir un determinado número de funciones de la conducción según su nivel de autonomía. El nivel de autonomía de un vehículo se encuentra definido de acuerdo con la escala *SAE*[50]. En la Tabla 2.1 se presentan los seis niveles de los que consta dicha escala.

Tabla 2.1: Niveles de autonomía de un vehículo. Escala *SAE*.

	Nivel	Nombre	Descripción narrativa	Ejemplo
El **conductor humano** monitoriza el entorno de conducción	**Nivel 0**	Sin automatizacion	La ejecución durante todo el tiempo de todos los aspectos dinamicos de la conducción, incluso mejorada mediante avisos y sistemas de intervención	Un vehículo conducido por un peaton, con o sin sistemas de asistencia a la conducción, i.e., un coche normal y un coche con sistema de frenada automatica de emergencia.
	Nivel 1	Asistencia al conductor	La conducción en ciertos modos especificos mediante un sistema de asistencia al conductor, que controle la direccion ó la aceleración/frenado usando información sobre el entorno y asumiendo que el conductor humano realiza el resto de tareas dinámicas de la conducción.	Un vehículo con control de crucero adaptativo y asistencia de parking (park assist).
	Nivel 2	Automatización parcial	La conducción se realiza en ciertos modos específicos mediante un sistema, que controla la dirección y la aceleración/frenado usando información acerca del entorno y asumiendo que el conductor hará el resto de las tareas de la conducción.	Un vehículo con sistema de conducción de crucero, i.e., mantiene el vehículo dentro del carril y acelera o frena si lo considera adecuado, pero que puede solicitar intervención humana en cualquier momento (el piloto automático de tesla).
El **conductor automático** monitoriza el entorno de conducción	**Nivel 3**	Automatización condicional	La conducción se realiza mediante un sistema automático que se ocupa de todas las tareas de la conducción, asumiendo que un conductor humano estará disponible para tomar el control en cualquier momento.	En este modo el coche decide cuando cambiar de carril, pero se espera que el conductor humano tome el control si algo no funciona según lo esperado.
	Nivel 4	Alto nivel de automatización	La conducción se realiza por un sistema automático, que realizará todas las tareas de la conducción, incluso si un conductor humano no puede tomar el control. En este nivel el vehículo puede conducir por si mismo durante todo el tiempo en condiciones normales, pero se aparcará sin poner a los pasajeros en riesgo si se encuentra una situación que no puede manejar y ningun conductor humano toma el control.	
	Nivel 5	Automatización total	La conducción se realiza por un sistema automático durante todo el tiempo y en cualquier condición que pueda realizar un conductor humano. En este nivel el vehículo puede conducir por cualquier carretera, en cualquier condicion con o sin conductor humano a bordo.	

[52]

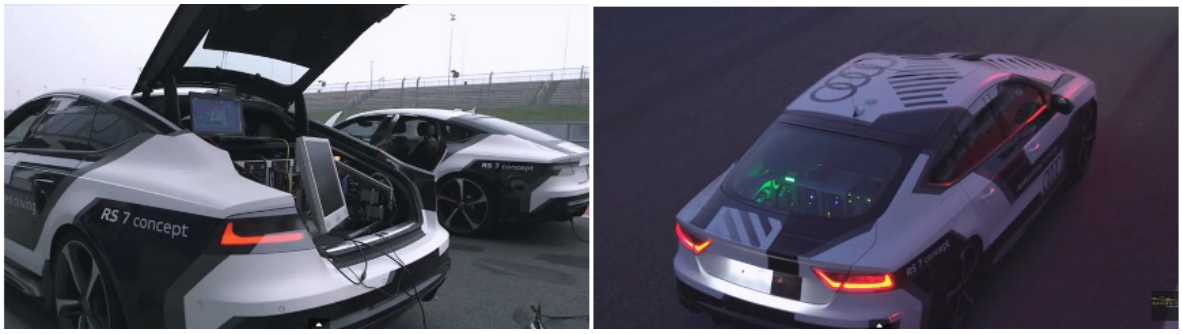
La historia relativa a los vehículos autónomos comienza a finales de la década de 1930. En 1939, Norman Bel Geddes[42] presenta la idea de los vehículos autónomos y su relación con la carretera.

Sin embargo, hasta 1980 no aparece el primer vehículo que circulaba gracias a un radar láser y visión artificial. Se trataba de un prototipo desarrollado[43] por la agencia de proyectos de investigación avanzada para la defensa de EE.UU. *(DARPA,* por sus siglas en Inglés).

En 1985, se publica un artículo (véase [2]) acerca del control de vehículos mediante técnicas de inteligencia artificial.

En 1995, según se indica en [3], Daimler-Benz y Ernst Dickmans preparan un vehículo con conducción autónoma para realizar el viaje de ida y vuelta Múnich - Odensee.

Ya en 2015, el fabricante de vehículos Audi afirma haber conseguido que su vehículo autónomo modelo RS7 recorriese el circuito de Hockenheim y alcanzase en él velocidades de hasta 240 km/h. En la Figura 2.7 se puede ver el aspecto de dicho vehículo.



AUTOGEFÜHL / [6]

Figura 2.7: Audi RS7 autónomo

Actualmente, el estado de Minnesota en EE.UU. se plantea modificar su legislación para regular la circulación de éstos vehículos en sus carreteras tal y como se indica en [4].

Éstos vehículos se encuentran compuestos en su mayoría por un vehículo convencional adaptado, que dispone de un ordenador de abordo que controla la operación del vehículo en todo momento. Además, disponen de sensores tales como sensores lidar (láser que permite calcular distancias), *GPS* y cámara para visión artificial. En estos vehículos es común que los sistemas de control, ya sea frenado y mantenimiento del vehículo en la calzada, se encuentren duplicados (lo que permitiría detectar errores) o incluso triplicados (de forma que se puede asumir un error en alguno de los sistemas).

El problema, al que los coches autónomos intentan hacer frente (véase sección 2.1), es la alta accidentalidad en las carreteras y el elevado número de víctimas derivadas de los accidentes de tráfico. Además, existe una creciente consciencia social con objeto de reducir el número de siniestros, así como el número de víctimas derivadas de éstos.

No obstante, la aproximación que se toma en este caso es diferente. Se parte de que un conductor humano no puede procesar la cantidad necesaria de información durante los instantes previos a un accidente. Y se asume que un agente inteligente artificial puede llegar en un futuro a asumir dicha cantidad de información y tomar mejores decisiones, dado que dispone de mayor capacidad de cálculo. La solución que ofrece el vehículo autónomo es dejar la tarea de conducir desde origen a destino a la máquina.

Se puede comprobar que un conductor humano habitualmente tiene que lidiar con áreas alrededor de un vehículo de las que no tiene ninguna información y que la conducción en velocidad fuera de poblado hace que los conductores no perciban un porcentaje de

las señales. Sin embargo, una máquina diseñada para no tener puntos ciegos, así como para procesar varias veces por segundo una imagen de situación, puede tomar mejores decisiones y por ende, conducir mejor.

Además, los vehículos autónomos permitirían una conducción más eficiente, puesto que podrían programarse para ser más respetuosos con el medio ambiente y transportarían sólo a aquellas personas que desean viajar al destino correspondiente.

En ultima instancia, la estandarización en el uso de estos vehículos puede llevar a que se realice un uso óptimo de las vías de transporte, puesto que éstas podrían admitir una mayor capacidad de vehículos, si se redujesen las distancias de seguridad necesarias para un transporte en condiciones de seguridad.

Sin embargo, los coches autónomos deben enfrentarse a los mismos problemas que los conductores humanos. En todo momento, el coche debe saber su posición tanto dentro de la carretera (i.e. ¿por qué carril circula?), así como la misma carretera en la que se encuentra (por ejemplo, la A-4). En este punto, conviene recordar que los sistemas de posicionamiento global permiten conocer su situación exacta con un pequeño margen de error, pero no eso no seria garantía suficiente de que estuviese en el carril correcto si la trazada del carril cambiase y el mapa no estuviese lo suficientemente actualizado.

Además, el vehículo autónomo debe ser consciente de los objetos que le rodean. En ningún caso se consideraría que éste realizase maniobras seguras si ignorase el estado de la carretera adyacente a su posición como vehículo con respecto a los demás, puesto que podría provocar colisiones y situaciones de riesgo al resto de vehículos y peatones que circulasen por la misma vía.

Con los puntos enumerados anteriormente se puede elaborar una composición de situación. Sin embargo, se necesita prever con bastante precisión que puede ocurrir en los instantes siguientes con el fin de realizar una conducción segura. Esto es, realizar una reducción de la marcha si hay peatones en las proximidades de un paso de cebra, o tan pronto se detecta un cambio drástico en las condiciones del tráfico. Un retraso en la detección de un embotellamiento puede llevar a consecuencias fatales.

En último lugar, el problema al que todos los conductores se enfrentan a diario es determinar que tienen que hacer en cada momento. Durante la conducción se deben tomar decisiones constantemente. Entre éstas se encuentran aquellas que tienen que ver con la dirección del vehículo (por ejemplo, mantenimiento/aumento/reducción de la velocidad,

encendido de la iluminación, selección del tipo de iluminación a emplear [p.ej. luces de cruce o luces de carretera],...), aquellas que tienen que ver con la circulación (por ejemplo, mantenimiento del vehículo centrado en el carril, mantenimiento de la distancia de seguridad, respeto al sentido de circulación, respeto a la señalización existente en términos de velocidad así como otras características, etc.) y por último, aquellas que tienen que ver con alcanzar el destino objetivo (conducción por la carretera adecuada, elección del itinerario óptimo, etc.).

A continuación, en las siguientes subsecciones se presenta una muestra significativa de los vehículos autónomos existentes.

2.2.1. Google Car / Waymo

En primer lugar, se aborda el vehículo autónomo de Google (ahora conocida como Waymo). Este vehículo cuenta con sensores que le permiten navegar de forma segura. Se ha diseñado principalmente pensando en los pasajeros, puesto que en este vehículo no hay conductor humano.

Dispone de multitud de sensores para ser consciente de los objetos que le rodean, así como de formas más redondeadas para facilitar que los sensores tengan áreas ciegas de reducidas dimensiones.

Actualmente, el proyecto sigue siendo un prototipo y se encuentra bajo pruebas para mejorar el rendimiento y la respuesta del vehículo. Puede consultarse más información al respecto del proyecto en la página web [5]. Cabe destacar, no obstante, que éste vehículo tuvo un accidente en circulación con un autobús tal y como se relata en [27].

THE GUARDIAN / [27]

Figura 2.8: Imágenes accidente de Google Car

Las pruebas (véase Figura 2.8) mostraron que se debía a un fallo por parte del piloto automático. El piloto automático analizó la información de la situación del vehículo y del tráfico a su alrededor y como resultado, comenzó la maniobra de incorporación al carril después de estar parado en la derecha. Éste no detectó, sin embargo, que el autobús no le cede el paso y se produce la colisión.

2.2.2. Ford

En segundo lugar, se aborda los avances de Ford en cuestiones relacionadas con el vehículo autónomo. En el año 2000, Ford publicaba una patente sobre el desarrollo de un vehículo autónomo en [28], que indica las partes de las que consta su prototipo de

vehículo. A día de hoy se encuentra en desarrollo del modelo autónomo Fusion Hybrid, que puede verse en la Figura 2.9. Según indica el fabricante, este vehículo va equipado con sensores LiDAR (es un tipo de radar), que permiten calcular distancias a otros objetos.

Figura 2.9: Ford Fusion Hybrid

2.2.3. Tesla Model S

Tesla ofrece desde 2014 el modelo Tesla Model S, que se ofrece como un vehículo de lujo con capacidades mejoradas. Dispone de serie de cámara delantera, radar delantero y sensores de sonar en los paragolpes delanteros y traseros. Con este equipamiento ofrece control de velocidad adaptativo, alerta por cambio involuntario de carril, piloto automático y asistencia de aparcamiento. Puede verse una imagen del coche en la Figura 2.10.

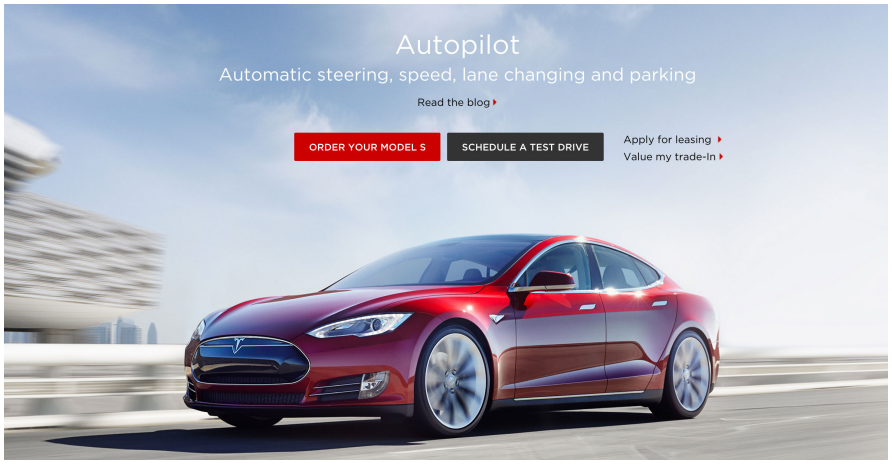


Figura 2.10: Tesla Model S

El piloto automático permite en determinadas situaciones realizar una conducción sin tener las manos en el volante. No obstante, el vehículo asume que el conductor humano se encuentra disponible para tomar el control, en caso de situación imprevista. Este vehículo también se vio involucrado en un accidente, en el que el piloto automático no detectó a tiempo un objeto y colisionó con él.

2.2.4. Mercedes Benz Clase S

Daimler Benz ha desarrollado soporte en la serie de vehículos Mercedes Benz Clase S, incluyendo Distronic Plus con Steering y Active Lane-Keeping. Estos sistemas permiten mantener velocidad de crucero adaptativa y alerta de cambio involuntario de carril.

La activación de los tres sistemas permite realizar al vehículo conducción autónoma en determinadas circunstancias, aunque se puede solicitar la intervención del conductor en cualquier momento.

El vehículo dispone de una cámara estéreo, así como de cinco radares. Puede verse una imagen de éste en la Figura 2.11.

Figura 2.11: Mercedes Benz Clase S

2.2.5. Continental Cube

Este vehículo autónomo presenta una alternativa a los desplazamientos privados por la ciudad. Al tratarse de un taxi, no necesita aparcarse cuando termina de realizar un traslado, y siendo un vehículo autónomo, “siempre” se encuentra disponible para el siguiente cliente. Puede verse una imagen de este vehículo en la Figura 2.12. Según indica su fabricante, también dispone de sensores láser para identificar si hay objetos o vehículos a su alrededor.

Figura 2.12: Continental Cube

2.3. Simuladores

En la presente sección se abordan los simuladores, el motivo de la existencia de éstos y sus diferentes variantes.

En primer lugar, se define[8] un simulador como un programa informático capaz de realizar simulaciones. Una simulación es una técnica de imitación o reproducción de procesos y/o eventos reales bajo unas condiciones de prueba en un tiempo determinado. La simulación es un proceso matemático complejo. Este proceso consta de un conjunto de reglas, relaciones, procedimientos operativos y variables.

Una simulación permite hacer experimentos y demostraciones. Los tipos de experimentos y demostraciones, que pueden realizarse en un simulador, ofrecen un abanico mayor de posibilidades, puesto que no están limitados por la peligrosidad ni el riesgo que conlleva realizarlos en realidad. Estos experimentos no requieren de un equipo sofisticado ni caro, por lo que su uso puede ser más extensivo. El uso de simulaciones permite ahorrar tiempo de desarrollo. Se fabrican sólo modelos probados y que responden a las características deseadas.

Si se desease probar un software sin simulaciones, se debería de disponer de equipamiento (tal como un vehículo de test) con el que realizar las pruebas, y éstas se encontrarían limitadas por el número de equipos de prueba que se encuentren disponibles al mismo tiempo (i.e., dos equipos no pueden hacer pruebas sobre el mismo equipo de test). Sin embargo, las pruebas virtuales permiten que varios equipos puedan trabajar en paralelo y disponer así de más capacidad para probar. De este modo, se puede reducir el tiempo de desarrollo y el tiempo necesario para las fases de desarrollo y pruebas de un prototipo se reduce debido a su uso. Además, los fallos ocurridos durante las simulaciones no implican consecuencias físicas, tales como daños al entorno y al equipo, y los experimentos pueden repetirse. Cada proyecto de desarrollo de nuevos sistemas contempla en todos los casos un periodo de pruebas con el fin de eliminar posibles fallos incluidos durante su desarrollo. En el desarrollo de sistemas complejos, el conjunto de pruebas que se deben realizar a menudo es extenso y minucioso. Además, después de cada desarrollo nuevo deberían volver a pasarse todas las pruebas para comprobar que la corrección de un fallo no ha introducido otro. Esta forma de proceder puede llevar a que el proceso de desarrollo se extienda tanto a lo largo del tiempo que suponga un coste muy elevado.

Con el fin de reducir los tiempos que lleva realizar las pruebas, así como los costes derivados en equipamiento, se utilizan simuladores para realizar parte de éstas. De hecho, no todas las pruebas pueden realizarse en los simuladores, puesto que existe en el desarrollo de sistemas autónomos y robots el concepto de diferencia simulación-realidad(*reality gap*). Esta diferencia es la responsable de que un sistema funcione de forma ideal en el simulador y sin embargo, falle de forma inconsistente cuando se aplica en un sistema real (por ejemplo, un coche en movimiento).

Dentro de los simuladores, éstos se encuentran divididos en categorías atendiendo a diferentes criterios. Si se observa solamente la naturaleza de su código fuente, estos pueden clasificarse en:

- De código cerrado. A pesar de que pueda desarrollarse código utilizando sus librerías, el código de éstas no se encuentra accesible, por lo que no se pueden realizar modificaciones al programa base. Por ejemplo, si en un navegador web faltase el botón “Atrás”, sin el código fuente no se podría añadir ese botón por parte de un tercero.
- De código abierto. También podría permitir desarrollar código usando sus librerías, pero a diferencia de la anterior, su código fuente si esta disponible para modificación. Este acceso permitiría incluir funciones nuevas o modificaciones a las incluidas para ofrecer mejor adaptación al problema a resolver.

Si se atiende al fin del software desarrollado, en este proyecto se clasifican los simuladores en tres categorías según su especificidad:

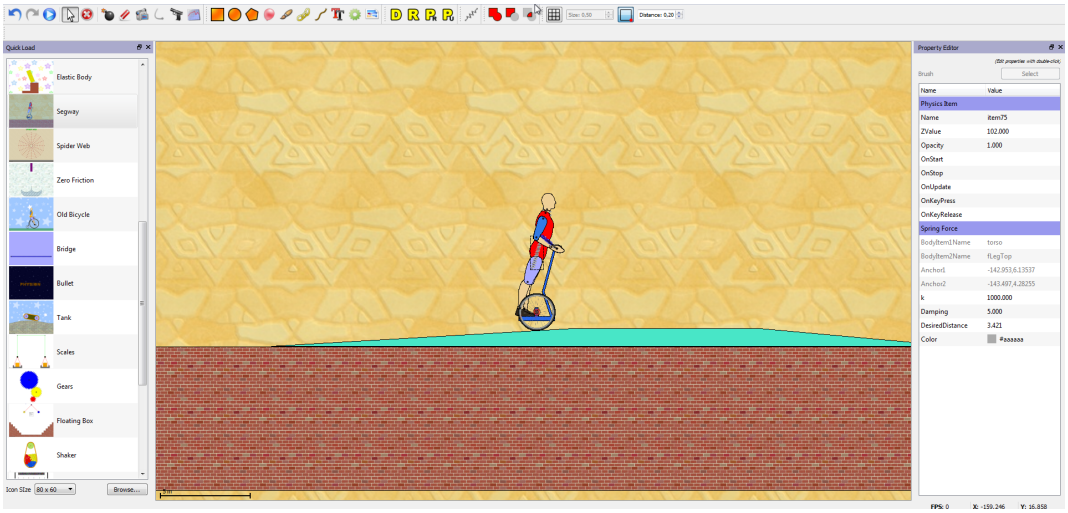
- Físicos.
- De robótica de propósito general.
- Específicos de coches.

2.3.1. Simuladores físicos

Los simuladores físicos permiten realizar experimentos con propiedades físicas, y su conjunto de primitivas suele ser bastante genérico. Se pueden usar tanto con fines educativos como de forma profesional.

- **Physion**

  Es un simulador físico creado en 2010 y ya va por su segunda versión. Puede verse una imagen del mismo en la Figura 2.13. Se encuentra disponible en [9]. Se utiliza en simulaciones interactivas con física y experimentos educativos. Se trata de un simulador de propósito general 100 % libre que permite trabajar con propiedades básicas.

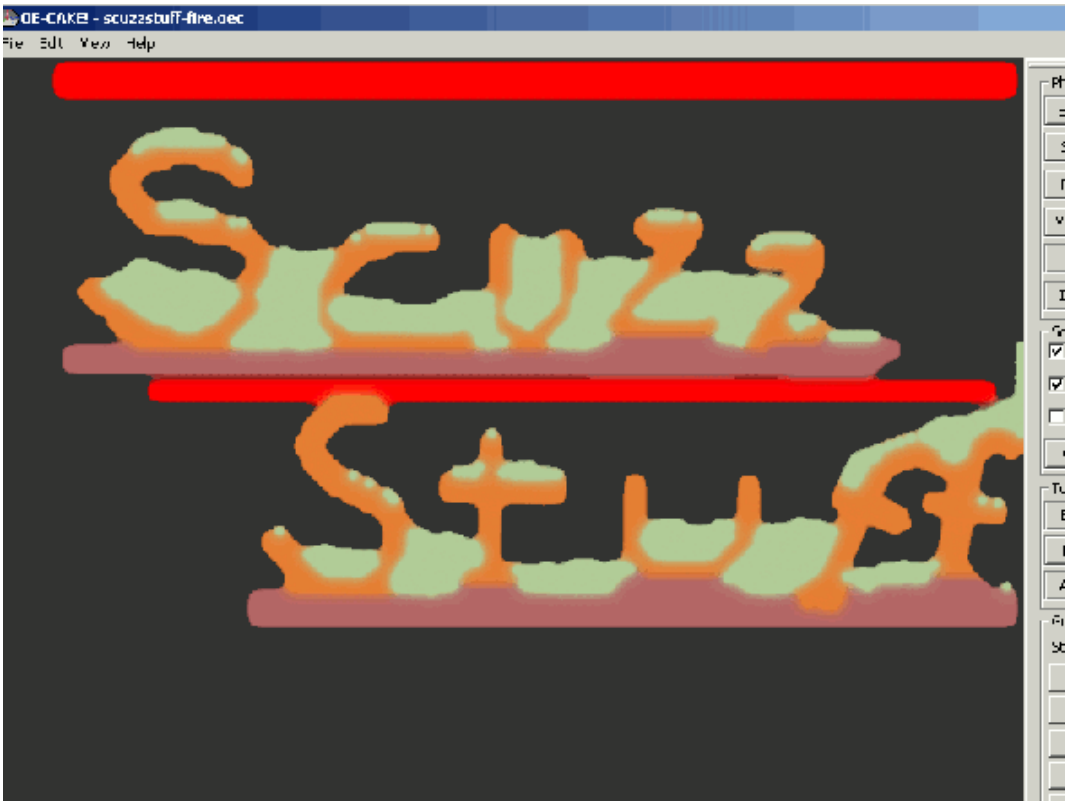


PHYSION.NET / [9]

Figura 2.13: Captura de Physion. Se muestra una simulación de un segway.

- **OE_Cake**

  Es un simulador físico, que permite tratar simulaciones físicas en las que intervienen también líquidos. Puede verse una captura de pantalla en la Figura 2.14. Se encuentra diseñado para uso educativo y permite visualizar los efectos que provocan elementos físicos en la simulación. Puede descargarse de aquí [10].

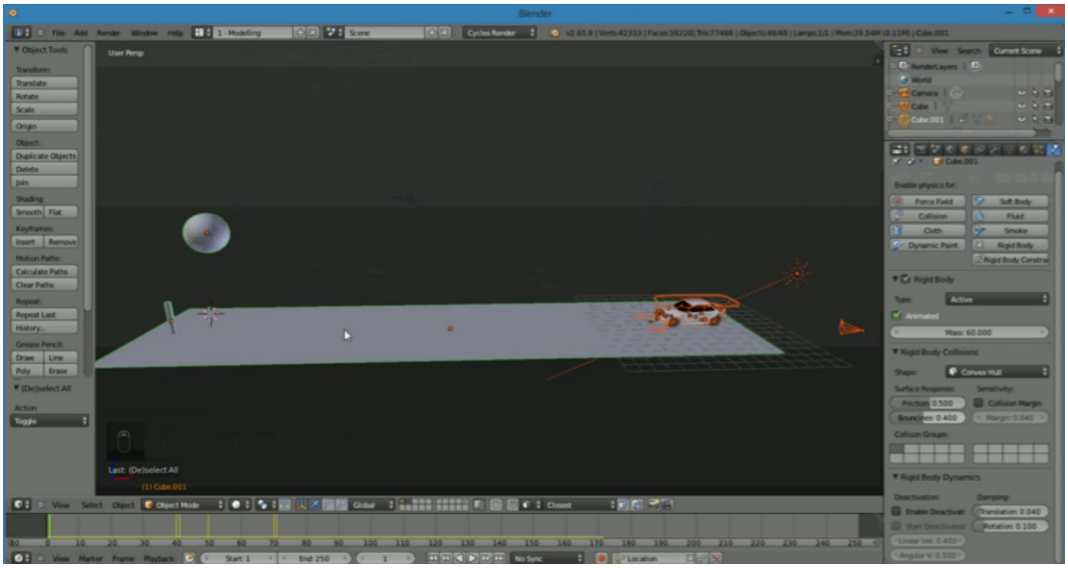


SCUZZSTUFF / [10]

Figura 2.14: Captura de OE_Cake

- **Blender**

  Blender no es estrictamente un simulador físico, a pesar de contar con un motor físico que permite realizar animaciones con diversas partes de la física (por ejemplo, mecánica clásica, mecánica de fluidos, óptica, etc.). Se trata de una aplicación de modelado de propósito general, que dispone de simulador físico. Éste dispone de capacidad para realizar animaciones que incluyan gravedad, física de partículas, líquidos, tejidos, colisiones y efectos de la luz. En la Figura 2.15 se puede ver una captura de pantalla de la aplicación. Más información acerca de la aplicación puede encontrarse en [11].



BLENDER / [11]

Figura 2.15: Captura de Blender. Esta figura muestra una escena de Blender con un experimento de colisión entre el vehículo y un poste vertical.

Se creó como una reescritura de una aplicación interna[53] para modelado *3D* de la empresa NeoGeo en 1995. En 2002 se liberó la primera versión de Blender. Sus creadores afirman que es de uso profesional.

2.3.2. Simuladores de robótica de propósito general

Los simuladores de propósito general permiten representar cualquier tipo de escena de forma virtual, teniendo un alto grado de libertad en las simulaciones. Sin embargo, su falta de especificidad implica que todos los elementos que formen parte de la simulación se tendrán que desarrollar por parte del usuario. Así pues, en el dominio de la conducción, ya se trate de ciudades o carretera, los elementos propios del dominio tendrán que construirse utilizando las primitivas ofrecidas por el simulador.

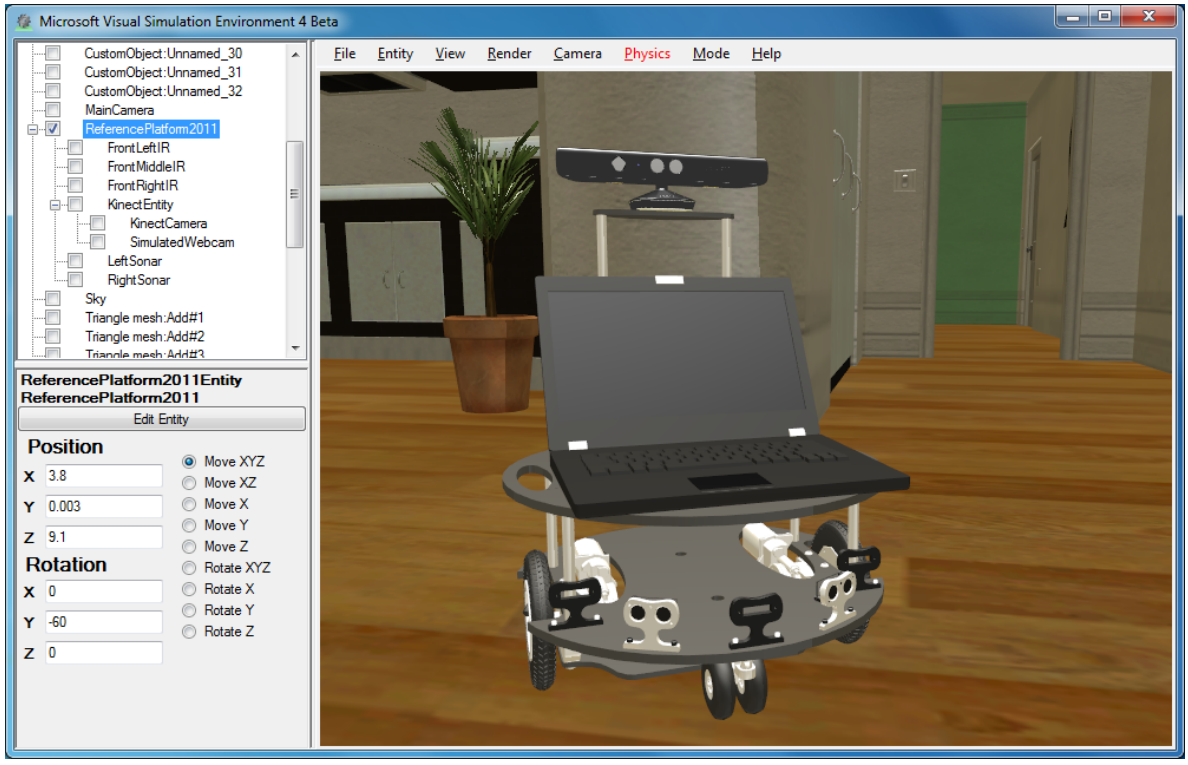
Los simuladores especializados en robótica permiten mostrar el comportamiento de robots en entornos simulados, de modo que pueda apreciarse la adaptación de su comportamiento para resolver un problema en un entorno lo más parecido posible a la realidad.

- **Microsoft Robotics Developer Studio**

  Se trata de un simulador de robótica de propósito general para uso aficionado, académico y profesional. Este simulador desarrollado por *Microsoft Inc.*, actualmente se encuentra fuera de desarrollo y su última versión es la 4.0.

  Dispone de un motor físico desarrollado por Ageia y usa el motor físico PhysX de *NVIDIA*. Solamente puede ejecutarse en la plataforma Windows y su lenguaje de programación principal es C#, aunque también pueden utilizarse otros lenguajes sobre la plataforma *.NET (ironpython, ironruby,...).* El principal formato para definir el entorno gráfico de la simulación es XML. Dispone de soporte para los siguientes tipos de robots: iRobot Roomba, LEGO MINDSTORMS NXT, MobileRobots Pioneer P3DX.

  La interfaz principal de usuario es de tipo gráfico y puede verse una captura de la aplicación en la Figura 2.16. Además, puede encontrarse más información en la página [12]. Soporta robots bípedos de pequeño tamaño.

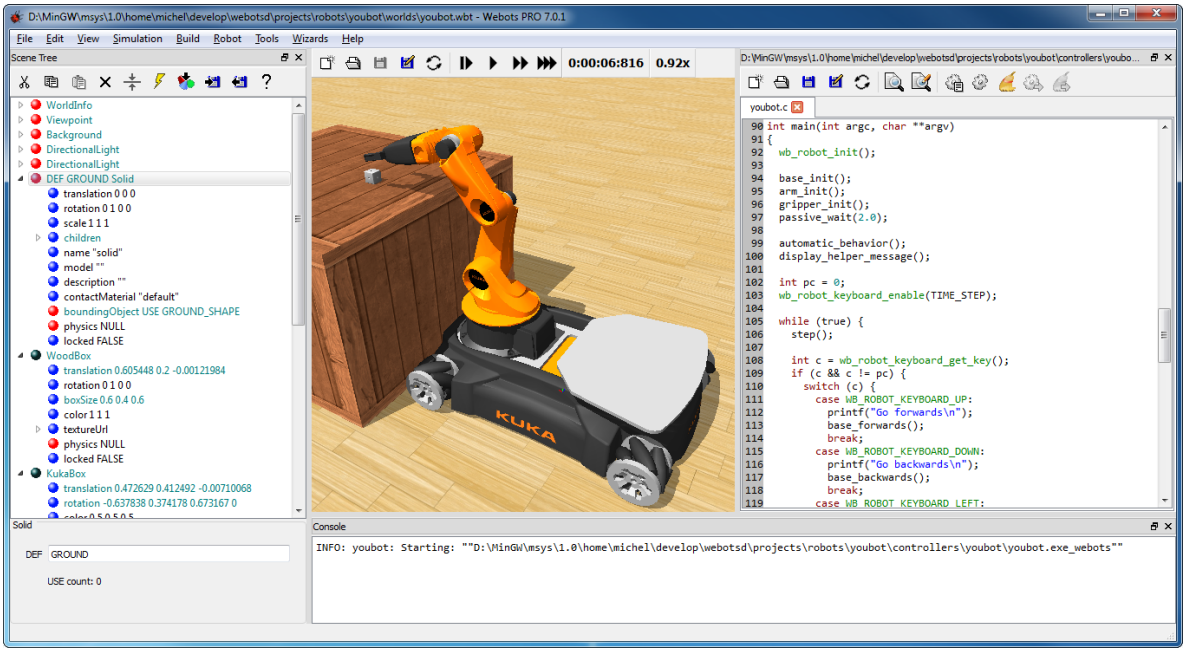


MICROSOFT / [12]

Figura 2.16: Captura de Microsoft Robotics Developer Studio

- **Webots**

  Se trata de un simulador de robótica de propósito general. Éste se utiliza para realizar simulaciones de robots y nuevos sistemas. Se encuentra desarrollado por la empresa *Cyberbotics*, su versión actual es 8.6.0 y tiene licencia propietaria. Este simulador dispone de OGRE como motor 3D y utiliza una versión adaptada de ODE como motor físico. Se puede ejecutar tanto Microsoft Windows, Linux, así como en MacOS. El principal lenguaje de programación, en el que programar las aplicaciones, es C. Aunque también ofrece soporte para utilizar C++, Java, Python y Matlab. Si se desea importar un modelo o entorno, se pueden utilizar los formatos WBT y VRML'97. Dispone de soporte para *ROS* (Robot Operating System). En la Figura 2.17 se muestra una captura de pantalla de la aplicación. Además, puede encontrarse más información al respecto en la página web [13]. También soporta robots bípedos de pequeño tamaño (*HOAP*, *Nao*).



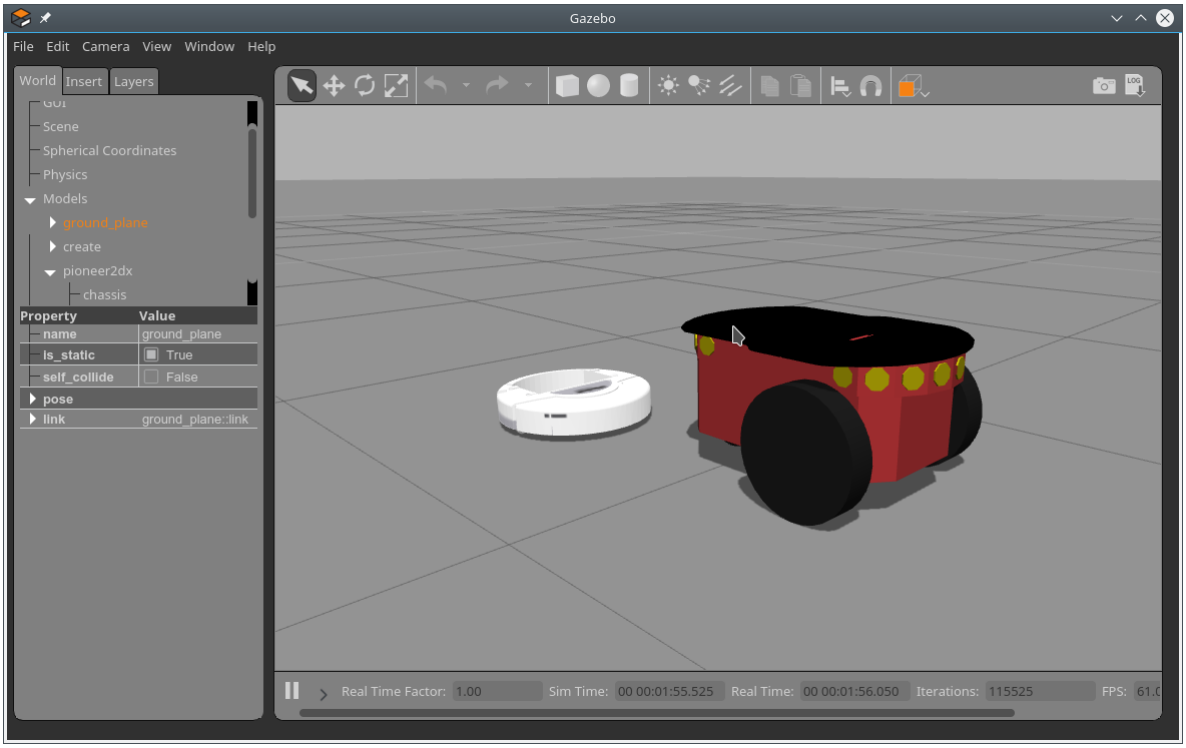
CYBERBOTICS / [13]

Figura 2.17: Captura de Webots

- **Gazebo**

  Gazebo se trata de un simulador de robótica de propósito general desarrollado por *Open Source Robotics Foundation* (OSRF). Se encuentra actualmente en desarrollo (v9.0.0) y licenciado bajo la licencia Apache 2.0 (*open source*). Al igual que *Webots*,

también utiliza el motor gráfico OGRE. Sin embargo, permite utilizar más de un motor físico además de *ODE* (como p.ej., *DART*, *Bullet* y *Simbody*). Este simulador puede ejecutarse en *Microsoft Windows*, *Linux* y *MacOS*. Sólo permite la programación de los controladores de robots en C++. Dispone también de soporte para *ROS*, así como *Player* y Sockets. La interfaz principal del usuario es a través de una interfaz gráfica y puede verse un pantallazo de esta en la Figura 2.18.



OPEN SOURCE ROBOTICS FOUNDATION

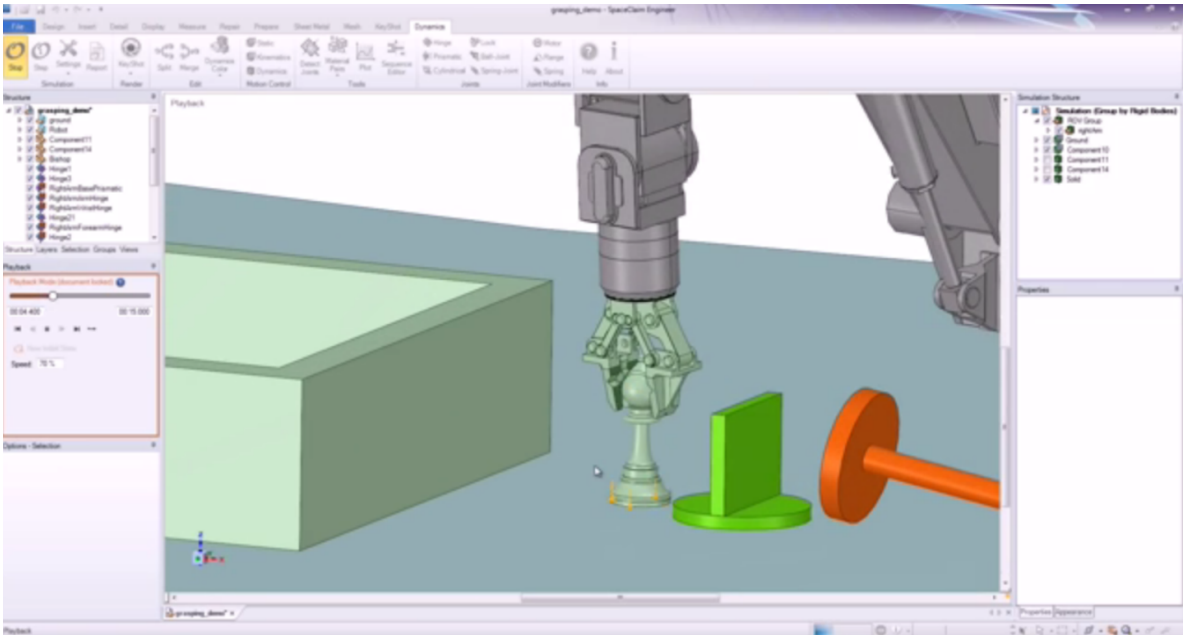
Figura 2.18: Captura de Gazebo

- **Dynamics for SpaceClaim**

  Dynamics for SpaceClaim [14] se trata de un simulador de robótica de propósito general de uso profesional. Se emplea para modelar experimentos con procesos complejos y desarrollar maquinaria especializada. Se puede encontrar una captura de pantalla de la aplicación en la Figura 2.19.

  Se puede considerar que esta aplicación permite realizar simulaciones sobre el comportamiento de los diferentes componentes de un sistema físico en una situa-

ción virtual. Por ejemplo, la aplicación puede simular el proceso de traslado de una carga mediante una grúa o la interacción entre dos engranajes a alta velocidad.



ALGORIX / [14]

Figura 2.19: Captura de Dynamics for SpaceClaim

2.3.3. Simuladores específicos de coches

Los simuladores de carácter específico se encuentran orientados a permitir las simulaciones dentro de un dominio concreto. Su alta especialización permite disponer de primitivas para la composición de escenarios más concretas y adaptadas al dominio del simulador. Estas ayudas agilizan el desarrollo de la escena y se busca que el simulador sea un elemento dinamizador que acelere las pruebas virtuales. Los simuladores específicos de coches en particular disponen de elementos que permiten simular con mayor facilidad escenas en las que intervienen vehículos de diferente tipo, así como elementos propios de la circulación.

- **Carsim**

  Se trata de un simulador específico para desarrollar nuevos sistemas y comprobar rendimiento de vehículos en desarrollo.

En su web[15], se indica que los peatones, así como otros actores en las simulaciones, se simulan de forma independiente al vehículo principal. Además, se especifica que se encuentran definidos mediante posición y orientación como mínimo, y que son detectables. Además, se permite controlar su movimiento, pero no se especifica con claridad si estos se encuentran sujetos a la física de la simulación.

Puede encontrarse una captura de pantalla en la Figura 2.20.

MECHANICAL SIMULATION / [15]

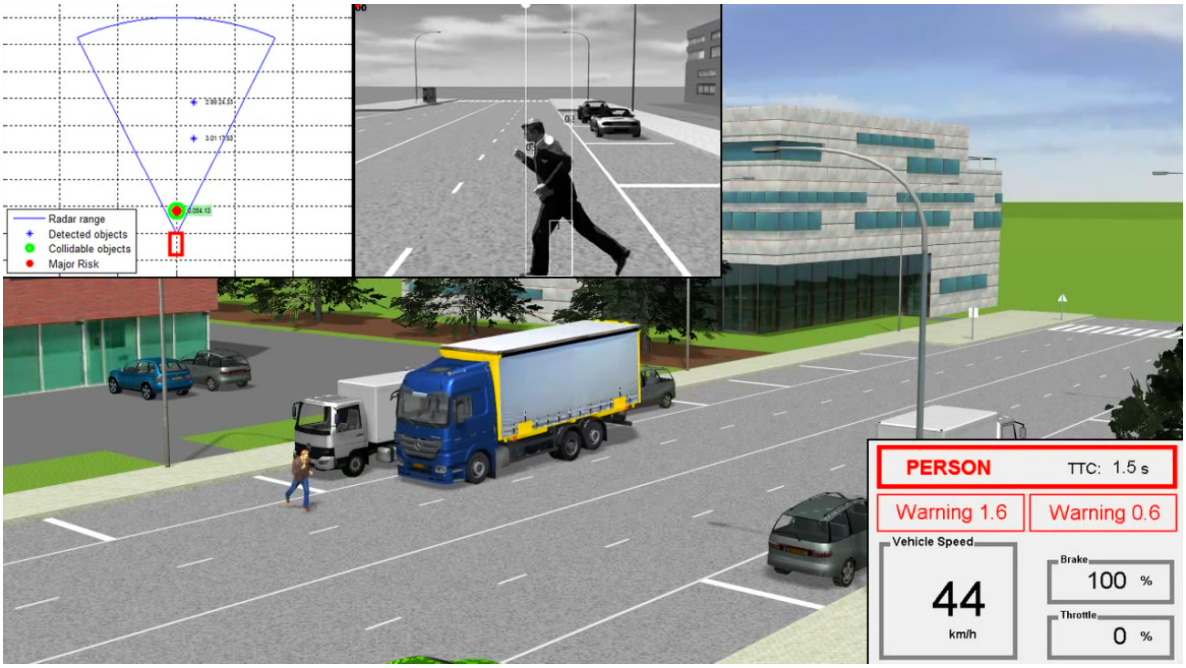
Figura 2.20: Captura de Carsim. La figura muestra una simulación de un vehículo en circulación siguiendo una secuencia de puntos de control previamente configurada.

- **PreScan**

  PreScan se trata de un simulador con soporte físico para desarrollar nuevos *ADAS* y elementos de seguridad activa.

  Tal y como se describe en su web[16], los peatones en PreScan se mueven con movimientos naturales. Sin embargo, tampoco se encuentra indicado con claridad si éstos están sujetos a la física del entorno. Por tanto, como mínimo los peatones se pueden controlar y detectar y se encuentran definidos utilizando su posición y su orientación.

  Se puede encontrar una captura de pantalla en la Figura 2.21.



TASS INTERNATIONAL / [41]

Figura 2.21: Captura de Prescan. Camión detectando cruce de peatón.

- **SCANeR Studio**

  SCANeR Studio [17] es un simulador específico de carácter propietario utilizado en la industria del automóvil. Se utiliza también para desarrollar nuevos *ADAS* y otros sistemas relacionados.

  Oktal, el fabricante de este software, menciona en su web que SCANeR Studio soporta el control de peatones y otros actores, pero no especifica si la física de la simulación también afecta a los peatones, y por tanto, son algo más que imágenes en movimiento.

  Se puede ver una captura de pantalla en la Figura 2.22.

OKTAL / [17]

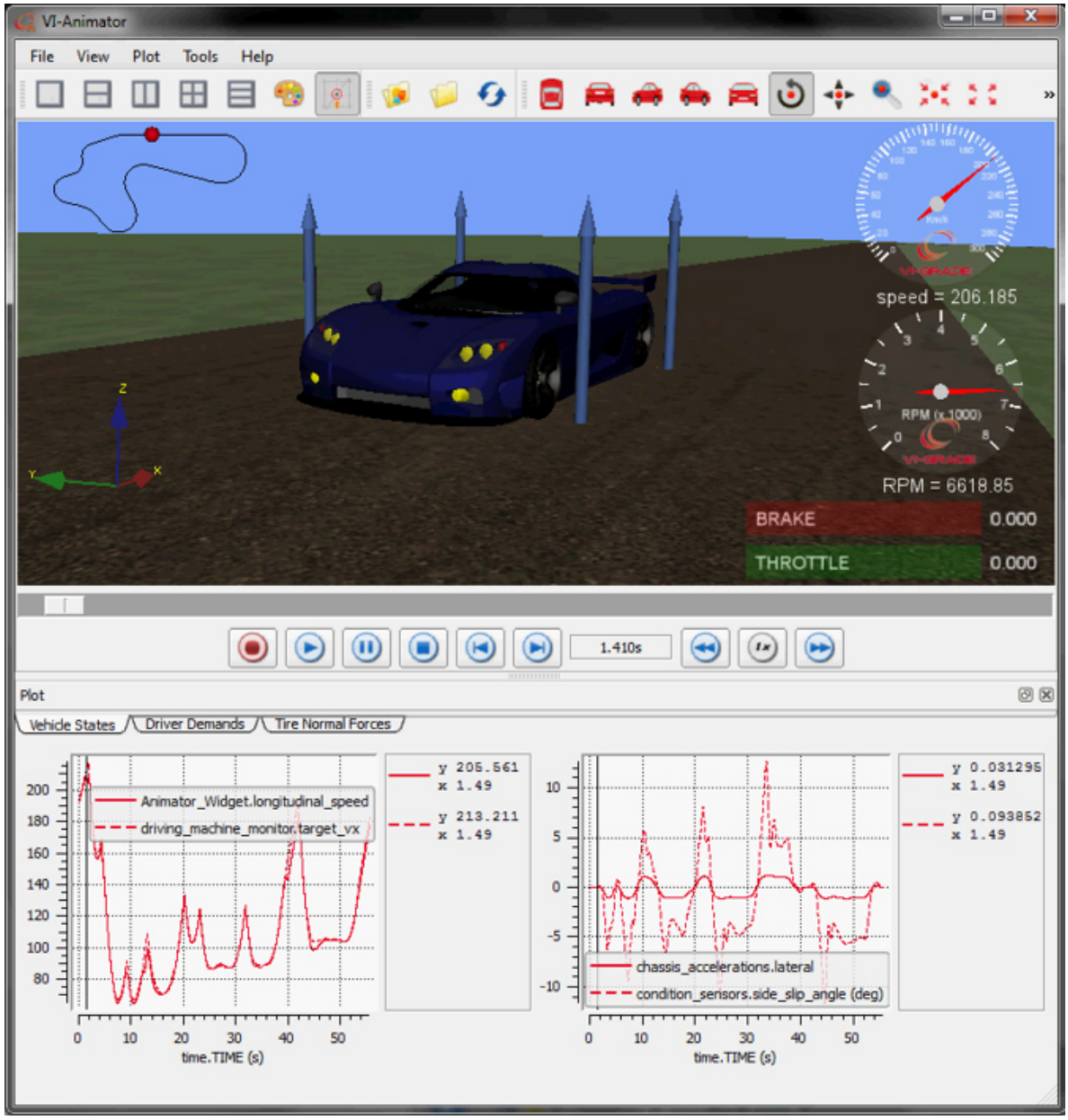
Figura 2.22: Captura de SCANeR. La figura muestra los sensores disponibles en una simulación de cruce.

- **VI-Real Time Car**

  VI-Real Time Car[18] es un simulador específico para desarrollo de nuevos ADAS y nuevos vehículos.

  En el caso de VI-Grade, se menciona de forma general que el software soporta algoritmos de prevención de colisión con peatones, pero no señala si el motor físico de la simulación también controla las fuerzas que se ejercen sobre los peatones.

  Se puede ver una captura de la aplicación en la Figura 2.23.



VI-GRADE / [18]

Figura 2.23: Captura de VI-Real Time Car

- **STI Sim Drive**

  Se trata de un simulador específico de coches que se puede emplear para desarrollar nuevos *ADAS*, así como nuevos vehículos y probar las respuestas del conductor.

  A continuación se muestra una captura de la aplicación (Figura 2.24).

STI SIM DRIVE / [40]

Figura 2.24: Captura de STI SIM DRIVE. Peatón cruzando calzada en simulación.

En su web[29] se indica que pueden utilizarse peatones para simular situaciones de la vida real, pero no específica con claridad si el tipo de los peatones implementados corresponde con objetos que pueden trasladarse siguiendo rutas predefinidas, o bien se trata de objetos con movimiento sujeto al motor físico que rige la simulación.

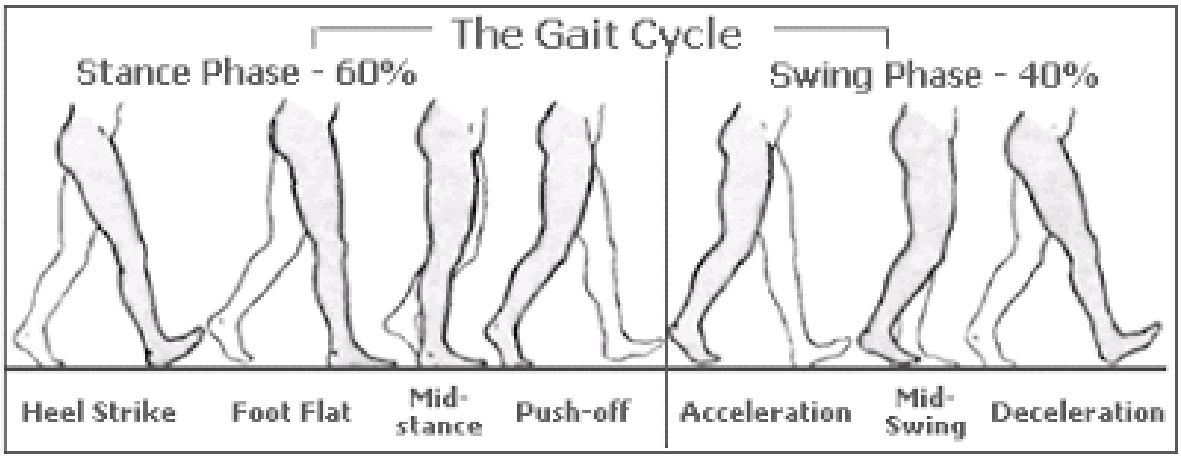
No obstante, la lectura de los manuales de STI Sim Drive no deja dudas respecto al tratamiento que se realiza con los peatones en la simulación. Estos se tratan como objetos detectables y cuya gestión se limita a desplazar un modelo 3D que cambia de forma cíclica a lo largo de la ruta indicada en la simulación, respetando la orientación indicada para cada punto de ésta.

2.4. Modelos bípedos de locomoción

Caminar, según queda definido en [8], es un medio rápido de locomoción terrestre, en el cual el cuerpo se mantiene erguido por encima del suelo mediante una serie de extremidades articuladas, las piernas. Puesto que éstas facilitan tanto apoyo como propulsión, la secuencia de movimientos que permiten caminar debe asegurar que el centro de gra-

vedad del ser humano siempre se mantiene dentro del área de apoyo. De lo contrario, la persona pierde el equilibrio y se cae.

En [7] se presenta una posible secuencia de las fases involucradas en el movimiento de caminar. Si se atiende a la Figura 2.25, se pueden apreciar 7 fases en las que se puede dividir el movimiento coordinado para dar un paso.



DR. M. C. / [7]

Figura 2.25: Fases del movimiento de caminar humano

Como se ve en la Figura 2.25, éstas fases además pueden agruparse en dos conjuntos, según corresponda a la pierna de apoyo o a la pierna en desplazamiento. A continuación se describen las fases correspondientes a la pierna de apoyo:

- Fase 1: Toque de tierra con talón (*Heelstrike*). Esta fase es la primera de aquellas correspondientes a la pierna de apoyo y con ella se realiza el primer apoyo sobre el suelo. La pierna en este punto se encuentra casi completamente estirada.
- Fase 2: Pie plano (*Footflat*). Acto seguido se apoya todo el pie sobre el suelo y el centro de gravedad pasa a encontrarse ya sobre el nuevo área de apoyo. Se realiza además una leve flexión de rodilla para aceptar el centro de gravedad.
- Fase 3: Posición intermedia (*Midstance*). En esta fase el centro de gravedad descansa aproximadamente en el centro del nuevo área de apoyo consolidando la posición de la pierna apoyada.

- Fase 4: Empuje (*Pushoff*). Una vez consolidada la posición anterior esta fase permite al individuo seguir andando, para lo cual propulsa al cuerpo hacia adelante para permitir un *heelstrike* por parte de la pierna contraria.

En este punto se describe las fases correspondientes a la pierna en desplazamiento:

- Fase 1: Aceleración (*Acceleration*). La pierna se flexiona ligeramente a la vez que se inicia el desplazamiento hacia adelante. Sin realizar ninguna flexión por parte de la pierna a desplazar no se realizaría ningún avance de ésta, pues el contacto con el suelo impediría tal movimiento.
- Fase 2: Balanceo (*Midswing*). En esta fase la flexión de la pierna en movimiento llega a su punto de flexión máxima. Se busca evitar un impacto entre el pie y el suelo, puesto que el movimiento de balanceo alcanza en este punto su velocidad máxima.
- Fase 3: Frenado (*Deceleration*). A lo largo de esta fase se produce el frenado de la pierna balanceada para iniciar una fase de *heelstrike* cuando se complete la apertura del paso.

En la Figura 2.26 se aprecia como las fases se encuentran agrupadas en dos grandes estados atendiendo a que la pierna sostenga el peso del cuerpo, o se encuentre desplazándose a una nueva posición de apoyo. Además, en cada fase puede verse la secuencia de estados por los que se transita hasta que se alcanza la siguiente fase. Conviene aclarar que al inicio del proceso se produce un desfase entre ambas piernas para permitir que se alternen los estados entre ellas sucesivamente.

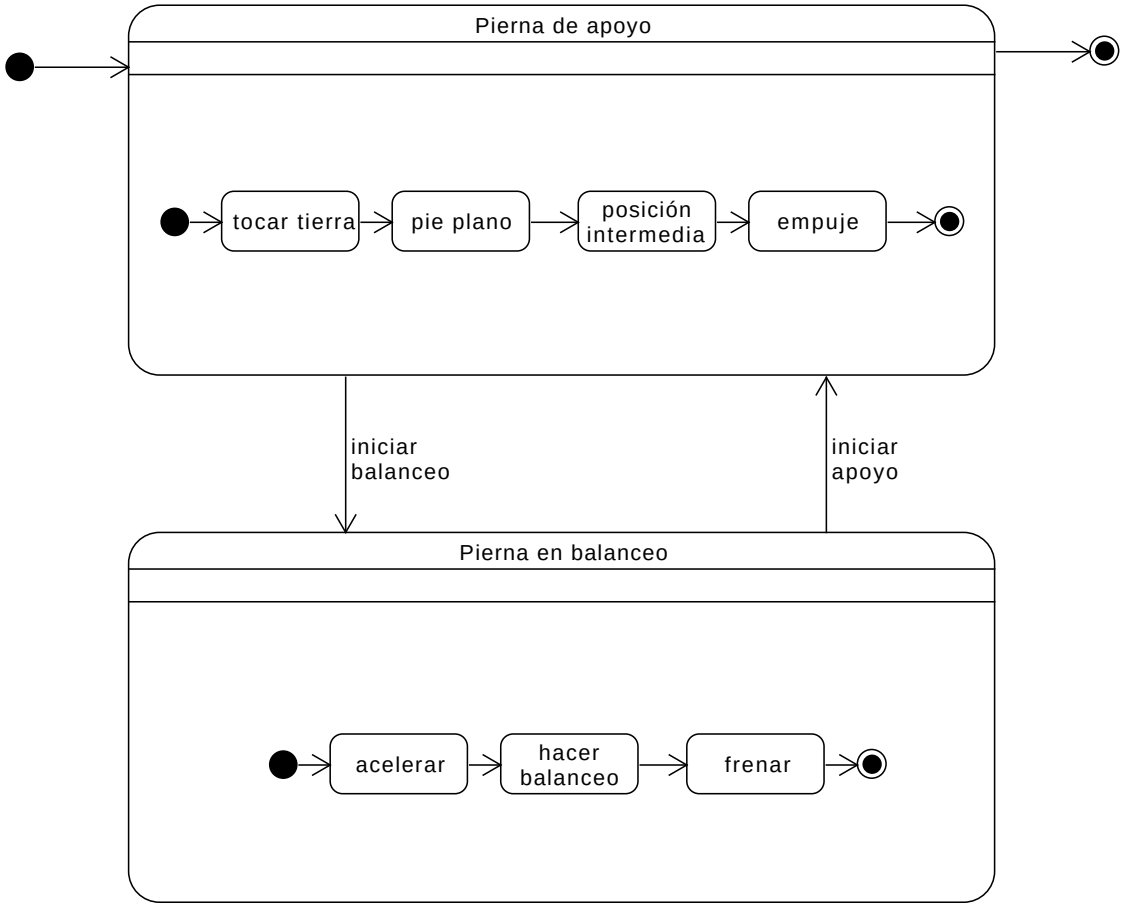


Figura 2.26: Diagrama de estado del proceso de caminar

Una vez visto el proceso mediante el cual el ser humano se desplaza, se van a describir un subconjunto de las técnicas empleadas en la materia para conseguir que robots bípedos sean capaces de caminar de un modo semejante.

En primer lugar se describe la técnica de Álvarez-Alvarez, Trivino y Cordón en [19]. La técnica descrita emplea máquinas de estado finito difusas y algoritmos genéticos.

El funcionamiento del modelo consiste en imitar lo más fielmente posible los movimientos realizados por las extremidades de un ser humano, de forma que un robot bípedo de características similares pueda andar replicando los movimientos registrados. Con este fin, un experto determina los estados de la máquina de estados finitos difusa y utiliza un algoritmo genético para aprender las reglas difusas y las funciones de membresía de

cada estado. Este aprendizaje se realiza analizando diferentes modalidades estudiadas a partir de diferentes personas.

El modelo se desarrolla en 2010 y se parte de que adquirir las ecuaciones diferenciales que determinan de forma exacta el proceso de caminar humano puede muy difícil o incluso imposible, proponiendo los sistemas difusos como alternativa viable a estos.

El objetivo último es extraer un modelo general que permita reproducir el patrón de movimiento que corresponde a dar un paso a un robot bípedo a partir de una generalización de un conjunto de partida. Este patrón no sólo tendría aplicaciones en la robótica, sino también en ámbitos de carácter médico.

Los resultados alcanzados con esta técnica muestran que se obtiene una serie de modelos con un ajuste significativo al modelo a imitar. Sin embargo, cuando se observa el ajuste de cada modelo con respecto al patrón de caminar distinto al del modelo a imitar, el error observado es bastante más elevado.

En segundo lugar, se tiene el modelo basado en ecuaciones diferenciales de Matos, V. y Santos, C. [55]. Este modelo describe los ángulos relativos entre los miembros de un peatón para cada instante como una ecuación diferencial no lineal. Por lo que cada articulación podría definirse de este modo. Sin embargo, solamente se modelan las articulaciones de la cadera ejes X, Y y Z y la rodilla, puesto que se asume que las articulaciones del tobillo pueden calcularse a partir de la suma de los valores de las funciones de cadera y rodilla, según sea el caso.

La sincronización entre las diferentes ecuaciones que componen el modelo se realiza mediante el uso de una tercera función, que hará las veces de generador de patrón. El objetivo de esta función consiste en generar valores, de forma que un subconjunto de éstos se repita a lo largo del tiempo. Este modelo es el que se toma de referencia en [54], con objeto de realizar una optimización de las ecuaciones definidas mediante la aplicación de programación genética.

2.5. Programación genética

La programación genética es una técnica de resolución de problemas basada en el mecanismo natural de la evolución, por el cual las soluciones más aptas para un problema

se reproducen más que aquellas que tienen una menor adaptación. Paradójicamente, las soluciones generadas mediante esta técnica estocástica convergen a la solución del problema a medida que las nuevas poblaciones reemplazan a sus antecesoras.

La técnica desarrollada por John Koza [20] actualmente es ampliamente utilizada para realizar aprendizajes y optimizaciones de soluciones a problemas de índole compleja, i.e. cuya resolución requiere de un proceso de búsqueda en el espacio de soluciones mediante un programa informático.

Se dice que es una técnica de resolución de problemas, puesto que el algoritmo propuesto realiza un búsqueda informada en el espacio de soluciones disponibles. El espacio de soluciones de un problema lo forman todas aquellas soluciones viables o no, que pueden producirse utilizando una determinada representación. En programación genética, el conjunto de soluciones disponibles lo forman todos los programas que pueden crearse para resolver un determinado problema. En este proceso de evaluar soluciones, es posible que algunas de éstas puedan no ser viables o aceptables. Sin embargo, las soluciones no viables deben también contemplarse, pues podrían aportar diversidad al proceso y facilitar de este modo su convergencia a una solución aceptable y óptima. Una posible forma para representar las soluciones en programación genética es emplear un árbol para caracterizar cada una de éstas. La representación en árbol de la solución permite seleccionar varias porciones de ésta o *genes* a la vez, que pueden tener sentido por sí solos. En las Figuras 2.27 y 2.28se muestran dos ejemplos de soluciones y su conversión en un árbol.

a) $\frac{2}{3} \cdot \phi_i$

b)

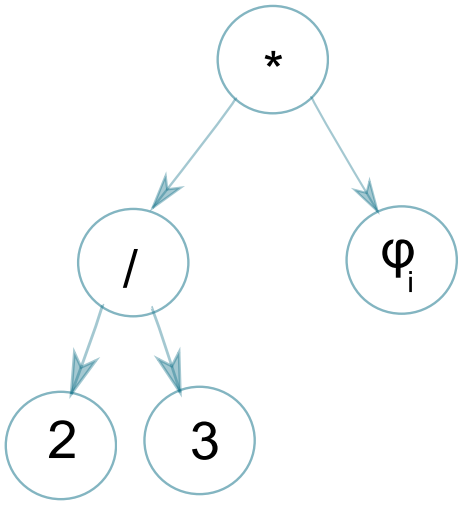


Figura 2.27: Ejemplo de conversión

a) $2 \cdot 2,24 \cdot \sin(\phi_i)$

b)

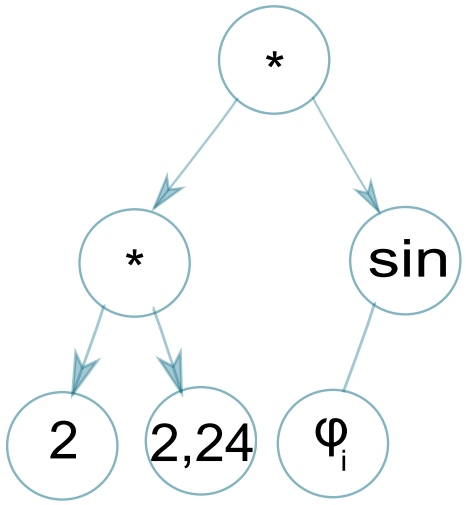


Figura 2.28: Ejemplo de conversión 2

Tal y como puede verse en las Figuras 2.27.b y 2.28.b, ambas muestran la representación de una solución o *genotipo*, y las Figuras 2.27.a y 2.28.a muestran su manifestación o *fenotipo*. Cada solución perteneciente a una generación se denomina *individuo*. Más adelante, en la sección 2.5.1, se presentan formas de crear la generación inicial, a partir de la cual se inicia el proceso de evolución.

Se dice que el proceso realiza una búsqueda, puesto que se procede a comprobar de entre todas las soluciones disponibles, cuál de ellas es la óptima, o dispone del mejor *fitness*, hasta que el proceso termina. Si la búsqueda no atiende a ningún criterio y simplemente busca de forma exhaustiva de entre todas las soluciones, se dice que la búsqueda es no informada o *ciega* (*blind search*). Realizar una evaluación exhaustiva de todos los programas creables para encontrar aquel que mejor resuelve el problema descrito, es una tarea que puede requerir muchos recursos y mucho tiempo, puesto que el espacio de soluciones de un problema puede ser muy amplio o incluso infinito. Por este motivo, se busca dirigir la búsqueda empleando algún criterio para no tener que revisar todos las posibles soluciones. Por el contrario, se denomina búsqueda *informada* o *heurística* a la búsqueda realizada, que descarta partes del espacio de búsqueda, teniendo en cuenta algún criterio de las soluciones generadas.

En la programación genética, el atributo que se tiene en cuenta para decidir no explorar un subconjunto del espacio de soluciones es la puntuación o *fitness.* Esta puntuación se asigna al individuo en función de la adaptación de éste al problema. Cada generación pasa por una fase de evaluación en la que todos los individuos se puntúan. Además, se dice que es una técnica estocástica debido a la naturaleza aleatoria que guía el proceso de búsqueda. Esto significa que las soluciones más aptas no van a incluirse siempre en el proceso de generación de nuevas soluciones, sino que éstas disponen de mayores probabilidades de ser incluidas (selección de las soluciones) y esto tiene importantes consecuencias. En primer lugar, este componente de azar lleva a que el proceso de búsqueda sea más resistente a quedarse estancado en máximos locales, puesto que la mejor solución hasta ese momento puede no emplearse en generar subsiguientes soluciones, y sí otras soluciones subóptimas que permitan salir del máximo local. En segundo lugar, esta característica puede llevar al proceso a no converger, puesto que pueden desecharse soluciones que conduzcan a la solución óptima. En último lugar, se pueden generar buenas soluciones en una determinada generación y que la siguiente generación no mejore dichas soluciones, por lo que se podría perder *fitness* en media a medida que se fuesen creando nuevas generaciones de soluciones, si las soluciones generadas fuesen peores que las de partida.

Se dice que el proceso se inspira en la teoría de la evolución biológica, puesto que en cada ronda se aplican al conjunto de soluciones (o *población*) los operadores genéticos adaptados a este dominio (i.e., se realiza una fase de cruce, otra de mutación y una inversión opcionalmente). La descripción de estos operadores se presenta a continuación :

- Cruce: Este operador va a crear dos soluciones nuevas a partir de dos existentes, combinando sus genes. Tal y como en la teoría de la evolución, se asume que se pueden obtener individuos con mejor *fitness* a partir de los genotipos de individuos con un buen *fitness* en la generación anterior. Sin embargo, esto no implica que después de cruzar dos individuos, los nuevos individuos vayan a disponer de forma automática de un mejor *fitness* en todos los casos.
- Mutación: Este operador parte de una solución y modifica uno de sus genes para generar una solución diferente. Al igual que en la teoría de la evolución, este operador, en las proporciones apropiadas, permite mantener la diversidad necesaria para que una población converja a la solución óptima.

- Inversión: Este operador simplemente altera el orden de dos genes en una solución. En determinados dominios, alterar el orden de los genes puede no ser posible o no tener consecuencias prácticas, por lo que puede dejarse fuera del proceso evolutivo.

El proceso y las fases que se siguen al aplicar esta técnica se describen a continuación:

1. Generación inicial. La generación inicial es la fase en la que se crean las soluciones con las que se comienza la evolución. Una buena generación inicial puede facilitar en gran medida que el algoritmo converja y encuentre una solución óptima en menor tiempo que si las soluciones generadas inicialmente son totalmente aleatorias. En la sección correspondiente se explica con mayor detalle las diferentes técnicas para generar individuos.
2. Evaluación. En esta fase se examinan las características del individuo y se le atribuye una puntuación o *fitness*. Este examen puede realizarse de forma interna o externa, dependiendo de si puede calcularse su *fitness* de forma sencilla en el propio programa o si debe acudirse a un programa externo para evaluar las características del individuo.
3. Selección. La fase de selección realiza una criba de los individuos disponibles para determinar cuales de los ellos pueden utilizarse para crear la próxima generación de individuos. En esta fase prima el fitness del que disponga cada individuo, pero no es indispensable tener el mejor *fitness* para estar incluido en este grupo. Se debe recordar que el proceso se denomina como estocástico, en parte porque los mejores individuos no tienen por qué ser aquellos con los que se genere la siguiente población. Más adelante se explican las diferentes técnicas existentes para llevar a cabo esta fase.
4. Cruce, Mutación e Inversión. En esta fase se crean los nuevos individuos pertenecientes a la nueva población. Posteriormente se explica la técnica que sigue cada uno, así como las variantes existentes de éstos.
5. Este proceso se repetiría volviendo a la fase 2 hasta que se alcanzase el número de iteraciones indicado o se encontrase un individuo cuyo *fitness* alcanzase el *fitness* objetivo propuesto.

En la Figura 2.29 puede verse una descripción gráfica del proceso :

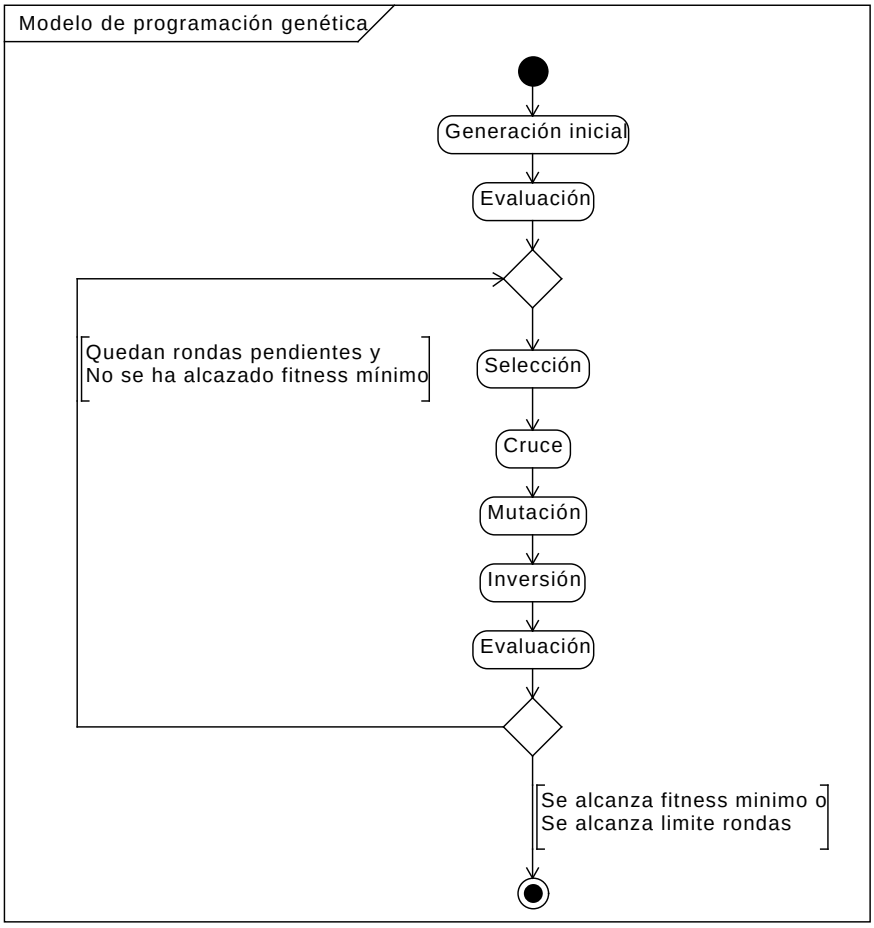


Figura 2.29: Diagrama de actividad ilustrativo del proceso evolutivo

2.5.1. Generación inicial

El algoritmo de programación genética requiere de una población inicial de soluciones que evolucionar para empezar a iterar. Por este motivo, la fase inicial de todo proceso evolutivo es adquirir una población inicial de soluciones. Estas soluciones pueden adquirirse de varias formas:

1. Generar el conjunto aleatoriamente: Las soluciones se generan atendiendo a unas reglas preestablecidas hasta que se alcanza el tamaño de población propuesto para el experimento. El hecho de incluir estas reglas introduce un sesgo en el proceso de

búsqueda que puede llevar a que el algoritmo pudiera no converger a una solución óptima. En determinados dominios en los que no se dispone de información acerca de soluciones con buen *fitness* puede ser una estrategia a tener en cuenta. la calidad de éstas se verá fuertemente influenciada por la adecuación del modelo de solución al problema descrito.

2. Incluir un conjunto de superindividuos en la población inicial: En este caso la forma de proporcionar las soluciones de partida es generar dichas soluciones con anterioridad a realizar el experimento e introducirlas al comienzo del proceso. Este método también introduce un sesgo en la búsqueda, puesto que las soluciones generadas a continuación, no disponen de un nivel de entropía tal como las del grupo anterior y tenderán a parecerse a las de partida. Sin embargo, el *fitness* de éstas será previsiblemente mejor, dado que están especialmente seleccionadas.

3. Realizar una combinación de las dos anteriores: Este método busca obtener lo mejor de los dos anteriores. A partir de un subconjunto de soluciones bastante aleatorio se busca tener mucha diversidad en la población, y sin embargo, las soluciones preseleccionadas con un buen *fitness* de partida permiten guiar la búsqueda en dirección al objetivo que se busca conseguir. No obstante, la forma de determinar los porcentajes de la composición es mediante prueba y error.

2.5.2. Selección

Con objeto de seleccionar individuos para crear la nueva población, se realiza un proceso estocástico de selección que tiene en cuenta la puntuación de los individuos o *fitness.* Sin embargo, como se ha adelantado en la sección anterior, este proceso no garantiza que un individuo con un *fitness* elevado con respecto a la media vaya a ser seleccionado para tomar parte en la generación de la nueva población.

Existen diferentes técnicas para realizar esta fase que tienen en cuenta esta característica y que se describen a continuación:

- Ruleta: La ruleta es una técnica de selección de individuos en la que se determina de forma aleatoria aquellos individuos con los que se generará la siguiente población. Sin embargo, la probabilidad de ser seleccionado crece conforme al *fitness* que tenga el individuo.[2]

[2]Hay que incluir fuentes a donde se ha utilizado y quién la inventó.

- Muestreo universal estocástico: Es una técnica similar a la ruleta para realizar selección de individuos. En la ruleta se sortea la posición de un puntero para identificar el individuo que es incluirá en la siguiente población. Ahora imagínese que se dispone de varios punteros equiespaciados y se sortea la posición del primero. Se seleccionan tantos individuos de una vez como punteros haya. Este método busca reducir el tiempo necesario para seleccionar individuos en una población de grandes dimensiones.

- Selección por truncado: Este método consiste en quedarse con un porcentaje de los individuos de la población actual ordenados por *fitness* para generar la siguiente.

- Torneos: En esta técnica se realizan tantos torneos como individuos vaya a tener la próxima generación. En cada torneo, se seleccionan aleatoriamente un número determinado de individuos y, de todos ellos, sólo el que tiene mejor *fitness* se utiliza. En función del número de individuos seleccionados para cada torneo, el algoritmo procede de una forma diferente.
  Si se utiliza un número de individuos bajo, la búsqueda se basa en exploración del dominio, puesto que soluciones con bajo *fitness* no se ven excluidas en el proceso de generación de nuevas soluciones.
  Por el contrario, si se utiliza un número de individuos alto, la búsqueda se basa en explotación de las mejores soluciones, dado que solamente soluciones con un buen *fitness* se emplearan en la generación de la nueva población. [3]

- Elitismo: Esta técnica consiste en seleccionar los N mejores individuos de la población actual para incluirlos en la población siguiente. Se busca mantener dentro del proceso evolutivo aquellas soluciones cuyo *fitness* se encuentra dentro del X % mejor, con el fin de seguir creando individuos a partir de éstas.

2.5.3. Operadores

A continuación se presentan los operadores existentes en la programación genética.

[3] Hay que incluir fuentes a donde se ha utilizado y quién la inventó. Incluir imágenes explicativas del proceso. Una imagen vale más que 1000 palabras.

2.5.3.1. Cruce

Una vez se encuentran seleccionados los individuos con los que generar la nueva población, hay que realizar el cruce de sus genes para obtener nuevos individuos y aquí también hay diferentes modos de realizar dicho proceso. A continuación se describen los dos modos más importantes para llevarlo a cabo:

- Cruce simple: Este cruce se basa en realizar una división en los genes de ambos individuos e intercambiar las partes. Con este cambio se consiguen dos individuos nuevos. Dada la naturaleza de la representación de las soluciones, es decir, en árbol, el cruce simple selecciona en ambos árboles un subárbol y realiza su intercambio. [4]
- Cruce multipunto: Se podría decir que el cruce multipunto es una generalización de la técnica anterior. En este caso, se llevan a cabo tantos cruces simples como se desee.

2.5.3.2. Mutación

La mutación tiene como objetivo modificar los genes del individuo para mantener la diversidad genética dentro de la población en cada generación. Las diferentes técnicas disponibles para llevar a cabo esta fase se describen a continuación:

- Mutación simple: La mutación simple consiste en cambiar un gen por otro aleatoriamente atendiendo a una probabilidad.
- Mutación en árbol: Esta mutación, propia de disponer de una representación en árbol, elimina el subárbol seleccionado y genera otro en su lugar según los parámetros especificados. [5]

[4] Incluir imágenes explicativas del proceso. Una imagen vale más que 1000 palabras.

[5] Hay que incluir fuentes a donde se ha utilizado y quién la inventó. Incluir imágenes explicativas del proceso. Una imagen vale más que 1000 palabras.

2.5.3.3. Inversión

La inversión consiste en reordenar dos genes consecutivos atendiendo a una probabilidad propuesta y genera un nuevo individuo en aquellos dominios en los que el orden de los genes sí tiene implicaciones en el comportamiento del individuo.

En la figura 2.30 puede verse un ejemplo del funcionamiento del operador de inversión sobre un árbol de expresión. El operador selecciona un nodo del árbol (el raíz) y selecciona dos nodos hijos consecutivos. Si el nodo no admite dos o más hijos, este operador no puede aplicarse. Por último, intercambia dichos nodos para producir un individuo distinto.

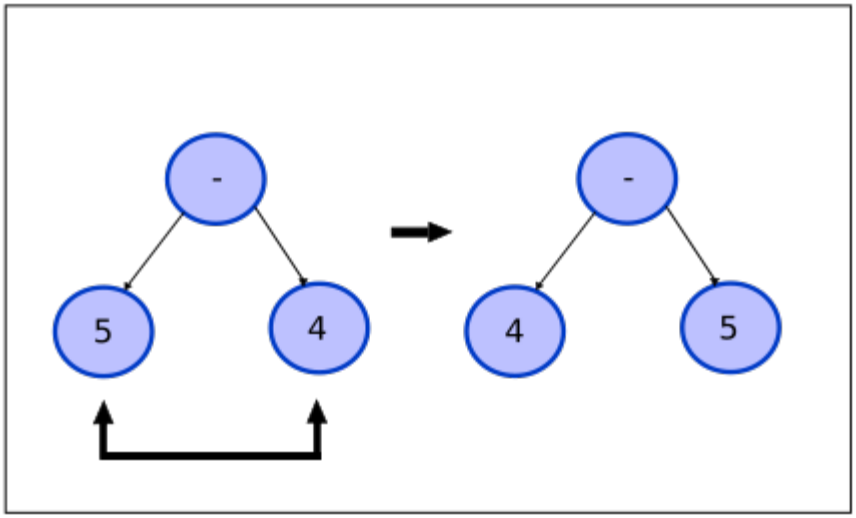


Figura 2.30: Ejemplo de aplicación del operador de inversión

En este ejemplo puede verse como se han intercambiado dos nodos hojas consecutivos de orden para producir otro individuo. Este cambio produce un árbol de expresión cuyo resultado es diferente al de origen.

2.6. Conclusiones

Los vehículos autónomos deben ser capaces de adaptarse al entorno cambiante. Sin embargo, su uso aún requiere de pruebas más exigentes, puesto que un accidente como el del coche autónomo de Google podría haber causado muchos mas daños, de no ser

por el conductor que se hizo con el control. Se hablaría en otros términos del accidente, si hubiese implicado a peatones.

En los simuladores específicos de coches que se han revisado no se ven indicaciones de que se usen modelos bípedos sujetos al motor físico de la simulación. El uso de estos simuladores es importante para realizar pruebas más complejas, en las que el movimiento del vehículo pueda verse afectado por colisiones con peatones. En estas condiciones, tras una caída del peatón, la detección de éste se complica. Esto significa que la inteligencia artificial (*IA*) podría determinar de forma equivocada que no hay obstáculos en el camino, mover el vehículo de acuerdo a esas condiciones y, en consecuencia, provocar mayores daños que la primera colisión. Por ejemplo, un vehículo puede golpear a un peatón en las cercanías de éste. Como consecuencia, el peatón cae y queda fuera del espacio analizable por los sensores del vehículo. Si la *IA* del vehículo determina que se puede mover sin restricciones, el movimiento podría provocar un atropello del peatón en el suelo.

Capítulo 3

Programación genética: Adaptación al dominio

3.1. Metodología de desarrollo

En el proceso de desarrollo de la herramienta de programación genética se han empleado diversas técnicas de desarrollo de software que han facilitado el mismo. A continuación se presentan cada una de ellas y la motivación subyacente para emplear dicha técnica en este proyecto.

3.1.1. Programación Ágil

La programación ágil describe un conjunto de valores y principios de desarrollo para hacer más dinámico el proceso de desarrollo de *software.* Desde la década de 1970 se buscaba realizar mejores desarrollos en menos tiempo, y con este fin surgieron muchos métodos (*rapid application process, unified process, scrum...*). En 2001 se reúnen diecisiete desarrolladores, entre los cuales estaban Jeff Sutherland, Ken Schwaber, Alistair Cockburn, Highsmith, Ken Beck. Este grupo de desarrolladores pone por escrito los principios de la programación ágil en el manifiesto ágil[21].

Esto no significa prescindir de los conceptos con fondo blanco de la tabla 3.1, si no que se priorizan los conceptos con fondo verde. Según indican sus autores, los principios en

los que se apoya la programación ágil pueden verse en la tabla 3.1, la cual se muestra a continuación.

Tabla 3.1: Principios de la programación ágil

Individuos e interacciones	sobre	procesos y herramientas
Software funcionando	sobre	documentación extensiva
Colaboración con el cliente	sobre	negociación contractual
Respuesta ante el cambio	sobre	seguir un plan

Además, sus autores especifican que todo lo mencionado en la tabla 3.1 es importante, aunque esta metodología pone el foco sobre los aspectos a la izquierda en dicha tabla.

En este proyecto se decidió emplear esta técnica debido a varias razones que se exponen a continuación.

- En primer lugar, se disponía de un software operativo adaptado a otro dominio. Este software precisaba de algunas operaciones de mantenimiento para poder utilizarse en este nuevo campo y poder ampliarse en el futuro. Además, éste se encontraba escrito en C++ con especificación de C++03[32] y necesitaba una actualización a la especificación de C++11[33].
- En segundo lugar, se consideraba que el número de cambios necesario era elevado y realizar una modificación utilizando una técnica similar a la metodología de desarrollo software “en cascada”[31] podría llevar a una situación aún peor. Sin embargo, la noción de que el código siempre funcionaba y de que se realizaban pequeños cambios asumibles mantuvo en todo momento el proyecto en marcha.
- En tercer lugar, se entendió el software como una herramienta para conseguir el objetivo final. El software debía ser una ayuda para conseguir un peatón que aprendiese a caminar, y como herramienta debía adaptarse lo mejor posible al trabajo a realizar. De este modo, se modificó el software para satisfacer las necesidades que del proyecto a lo largo de todo su ciclo de vida de forma reactiva.

- En cuarto lugar, emplear esta técnica permitió tener una mayor flexibilidad ante los cambios que el código debía admitir. Estos cambios eran fruto de las necesidades cambiantes y de requisitos no previstos en el momento del arranque del proyecto. En un primer momento habría sido muy difícil elaborar detallados planes de desarrollo que contemplasen los requisitos que se necesitarían a lo largo de todo el proyecto.
- En quinto y último lugar, se disponía de la libertad necesaria para acometer las modificaciones necesarias a la aplicación sin estar sujeto a licencias de uso externas (p.ej. *EULA* de Microsoft[34]). A posteriori, la modificación del software inicial hasta el software del que se dispone en la actualidad llevó asociado un proceso de aprendizaje valioso.

Sin embargo, resultaba muy sencillo ver los requisitos como una sucesión de cambios individuales, que, ordenados de forma conveniente, se podían afrontar mediante un esfuerzo de desarrollo mínimo. Esa sensación de facilidad condujo a realizar complejas transformaciones en algunos casos con bastante naturalidad. A esto último también ayudó utilizar el método Mikado, que se trata más adelante.

3.1.2. Programación extrema (XP)

La programación extrema o *extreme programming* (XP, por sus siglas en Inglés) es una metodología de desarrollo ágil para crear y mantener software. Esta metodología (véase [22]) incluye una serie de técnicas para mejorar el proceso de desarrollo de software y hacerlo más eficiente y estable. Sin embargo, la metodología esta pensada para un equipo de desarrollo, por lo que fue necesario adaptarla para que fuese útil para un sólo desarrollador. En esta adaptación se tuvieron en cuenta las siguientes técnicas descritas:

- Método de planificación (planning game): Esta técnica consiste en estimar, priorizar y reajustar una planificación para que sea llevada a la práctica con éxito.
- Desarrollo dirigido por pruebas (TDD): El desarrollo dirigido por pruebas busca que los programadores se plateen posibles usos del software y desarrollen pruebas para verificar ese funcionamiento. De este modo, las pruebas cumplen una doble función. Por un lado, permiten especificar el modo de uso de una funcionalidad y por otro, verifican su correcto funcionamiento. Todo el software que se desarrolló, después de incluir las pruebas automáticas, llevó aparejadas las correspondientes pruebas necesarias para verificar que el programa cumplía con los requisitos establecidos.

- Entregas rápidas: Las entregas rápidas llevan asociado que el software siempre se encuentra en buen estado, se encuentra listo para entregarse al cliente y que los ciclos de desarrollo-entrega son de corta duración. Esto determina que el *software* desplegado sea un pequeño incremento (i.e., pocas modificaciones o *software* nuevo) respecto de la ultima versión disponible.
  En el desarrollo de este proyecto se intentó minimizar la introducción de errores en la base de código de la aplicación. Los defectos que podía tener la aplicación tardaban mucho en manifestarse en una ejecución normal y su depuración no podía realizarse mediante logs (no se podía aplicar un nivel de log suficiente para ello) o utilizando un entorno de desarrollo integrado (o *IDE*, por sus siglas en inglés). Esto se debe a que las ejecuciones de la aplicación podían durar semanas o incluso meses (la mayor duró mes y medio). Se buscaba que las ejecuciones durasen lo máximo posible, porque se trataba de un factor que limitaba los experimentos. En algunos casos, no se alcanzaba la ronda objetivo en caso de no llegar al fitness mínimo para detener el experimento, porque se había producido un fallo que detenía la ejecución del experimento. Por este motivo (minimizar la introducción de errores), el desarrollo se hizo en incrementos cortos. Desarrollar de este modo permite detectar con mayor facilidad el origen de un fallo. Si éste esta relacionado con código bajo desarrollo, se necesitan inspeccionar sólo las líneas de código asociadas y su entorno para hallar el error que lo provoca.

- Diseño simple: El principio de diseño simple busca que el programador implemente el diseño más sencillo en el caso de encontrar varios diseños posibles, para facilitar tanto su desarrollo como su posterior mantenimiento o extensión. Además, este principio también indica que el programador debe revisar su diseño, para evitar un posible aumento de la complejidad en posteriores extensiones. Asimismo, el diseño debe revisarse para evitar que se produzcan código redundante.
  En este proyecto el diseño simple se adoptó para mantener la solución más simple y de este modo, facilitar el posterior mantenimiento del mismo.

Las técnicas que se describen a continuación no forman parte estrictamente de la programación extrema, pero se encuentran relacionadas con ésta y se emplearon en los procesos evolutivos que afrontó la base de código de partida. Estas técnicas son:

- Desarrollo dirigido por comportamientos/funcionalidad (BDD): La programación orientada a comportamiento es una técnica de desarrollo derivada de la programación orientada a pruebas, en la que se desarrollan primero las pruebas necesarias para

definir la funcionalidad y posteriormente, se desarrolla el código que satisface las pruebas desarrolladas. El proceso que se sigue en esta técnica es el siguiente:

Tabla 3.2: Fases del desarrollo orientado a funcionalidad

Definición de la funcionalidad
Implementación de la funcionalidad
Fase de refactoring

A continuación se describen las fases de que consta (véase Figura 3.2):

- Fase 1: Desarrollo de las pruebas que definen la funcionalidad. En esta fase, las pruebas deben probarse con código simple para verificar el correcto funcionamiento de éstas. Además, todas las pruebas deben fallar al menos una vez para comprobar que la prueba puede detectar algo.
- Fase 2: Desarrollo del código que la implementa. En esta fase se desarrolla el código y se modifica éste, hasta que todas las pruebas automáticas pasan.
- Fase 3: Se lleva a cabo la fase de *refactoring*, en la cual el código se revisa y modifica para eliminar duplicidades y simplificarlo en la medida de lo posible.

En este proyecto se empleó esta técnica con el fin de que las nuevas especificaciones pudiesen verificarse a lo largo del proyecto. De este modo, se podría comprobar que especificación no se está cumpliendo o produce un error de funcionamiento. Esta necesidad viene motivada por la dificultad existente en este campo para validar el correcto funcionamiento de una determinada característica o funcionalidad meramente haciendo una ejecución del programa (i.e. durante ésta parece que se produce una evolución). Además, se necesitaba una fiabilidad alta en el programa, puesto que se realizaban ejecuciones de larga duración (p.ej. algunos experimentos llegaron a las tres semanas de computo), y un error podía suponer tener parar el experimento y comenzar de nuevo.

- Programación de pruebas automáticas para código existente (TAD): Este proceso consiste en incorporar pruebas automáticas en aquellos programas ya existentes en los cuales dichas pruebas no existen. Principalmente, las modificaciones que deben realizarse en el código fuente de la aplicación para incorporar las pruebas

automáticas son numerosas y deben realizarse de forma metódica. El objeto de la modificación debe ser estrictamente la incorporación de dichas pruebas, y no se debe modificar el comportamiento de éste. En este proyecto se empleó esta técnica con el fin de introducir pruebas automáticas en aquellas partes existentes, en el momento de la introducción de las pruebas automáticas para mejorar la fiabilidad del código. Si bien es cierto que las pruebas automáticas no garantizan la ausencia de errores, sí permiten detectar aquellos que se han producido anteriormente[1]. Esta detección precoz permite eliminar los fallos durante el desarrollo y producir un código de mayor fiabilidad.

3.1.3. Método Mikado

A continuación se explica el método Mikado, que es una técnica que se empleó a mitad del proyecto para poder introducir *tests* automáticos que permitiesen verificar la aplicación de programación genética en cualquier momento.

El método Mikado (véase [23]) es un método estructurado para realizar cambios importantes a un código complejo. Es un método gráfico que permite seguir los cambios y precondiciones visualmente. También puede utilizarse en otros contextos de forma más general que no involucren modificar código.

En la formulación del método Mikado se hace uso de algunos conceptos, que pasan a desarrollarse a continuación:

- Meta: Es el objetivo final que se persigue conseguir con la aplicación de este método, p.ej. eliminar del código de la aplicación una dirección web como un literal.
- Precondición: Se trata de cada una de los condiciones imprescindibles, sin las cuales no se puede alcanzar la meta. En este caso, sustituir la dirección por un parámetro.
- Parte afectada: Se trata de la parte del sistema que muestra cambios cuando comprobamos una precondición. Por ejemplo, en este caso el módulo de descarga de páginas web.

[1] Siempre y cuando pueda desarrollarse una prueba para verificar que el código cumple la condición.

- Deshacer: Se trata de la fase en la que deshacemos los cambios aplicados, cuando se ve que se producen fallos al comprobar una precondición, p.ej., restituir el literal con la dirección web.

La principal característica del método es que es una técnica gráfica. Además, se utiliza para mantener la consistencia del sistema, en este caso, una base de código. Normalmente, los nodos del diagrama se ordenan de abajo a arriba acorde a su importancia (i.e. la meta, siendo el más importante, se sitúa en la parte más baja), no suele presentar colores ni tampoco notas.

Sin embargo, esta técnica se adaptó a este proyecto en varios aspectos. En primer lugar, se ha ordenado el nodo más importante (i.e., la meta) en la parte superior. Se incluyen colores para ilustrar el resultado de comprobar una precondición. De este modo puede verse con mayor rapidez qué precondiciones funcionan. Por último, se asocian notas aclaratorias por nodo, de modo que se puede saber la causa de un fallo.

A continuación se presenta en la figura 3.1 una comparación entre el grafo Mikado estándar y el personalizado para este proyecto.

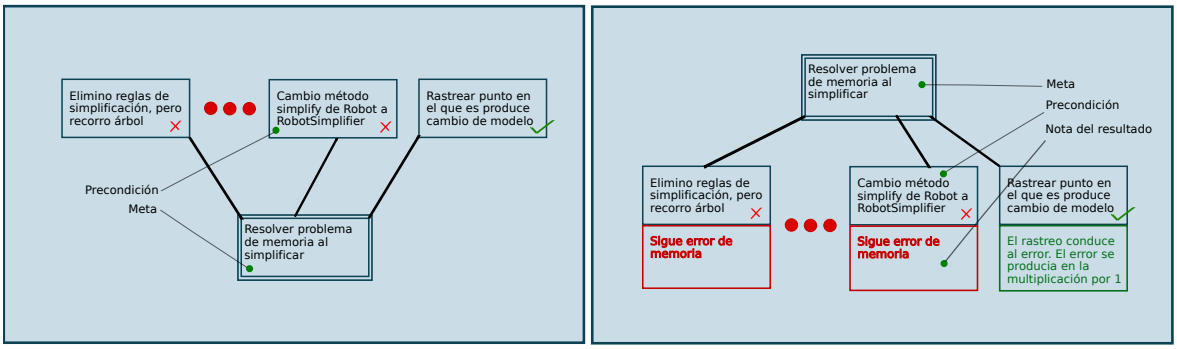


Figura 3.1: Comparación gráfica entre grafo estándar mikado y personalizado

Según sus autores, Ellnestam y Brolund, el método Mikado se ha aplicado con éxito al hacer cambios para mejorar la arquitectura de una aplicación, realizar *refactoring (*mejorar el código sin alterar la funcionalidad) en partes de ésta y realizar cambios dentro de una aplicación en producción. En esta última categoría entran tanto la inclusión de nuevas funcionalidades, como la corrección de fallos escurridizos.

Entre los beneficios que se obtienen al emplear este método, se pueden listar los siguientes:

- Modificar una base de código amplia de forma compleja, sin riesgo a realizar cambios que pueda suponer perder la trazabilidad (i.e., perder la pista en caso de marcha atrás). Incluso cuando este proceso dure semanas o meses.
- Es un diagrama sencillo y modificable que ofrece el estado de la modificación de forma rápida.

La aplicación del método se realiza siguiendo el diagrama que se presenta en la figura 3.2

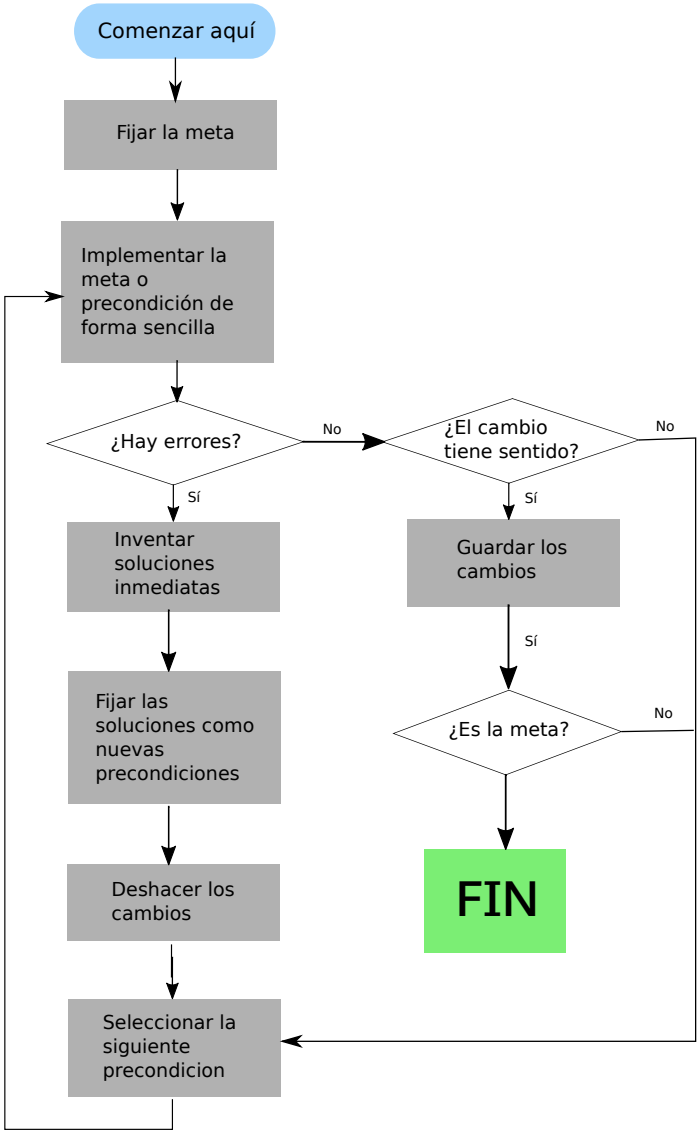


Figura 3.2: Modo de aplicación del método Mikado

Tal y como se comentó al inicio de la sección, esta técnica se aplicó a mitad del proyecto, cuando se decidió incluir tests automáticos. Se trataba de descartar errores en el código. El estado inicial del código de la aplicación utilizada disponía de muchas dependencias entre sí y no se encontraba preparado para soportar la inclusión de tests. Además, a medida que se fueron realizando experimentos la hipótesis de que hubiese errores en la aplicación fue volviéndose más probable. La forma de mantener la base de código estable y compilando en todo el proceso se consiguió mediante la aplicación de este método.

3.2. Desarrollo software de la aplicación de PG

En esta sección se describe la arquitectura del software desarrollado para realizar los experimentos de programación genética y las técnicas empleadas para desarrollarla. Además, se incluyen diagramas para explicar tanto el funcionamiento como la estructura de la aplicación. El detalle de los componentes y paquetes esta inspirado en el estándar ESA-Lite. Esta adaptación del estándar ESA para proyecto de software de pequeño tamaño no se utiliza de forma completa puesto que se trata de un proyecto muy pequeño y llevado a cabo por un equipo reducido (una persona), por lo que se ha buscado adecuar la documentación a generar a la dimensión del proyecto y al equipo.

En primer lugar, se presenta el diseño de la arquitectura software empleada, para mostrar una vista abstracta y general de la estructura que tiene el software. A continuación, se incluye una descripción general de los diferentes componentes que integran la aplicación para mostrar las diferentes funcionalidades que tienen y sus correspondientes estructuras internas.

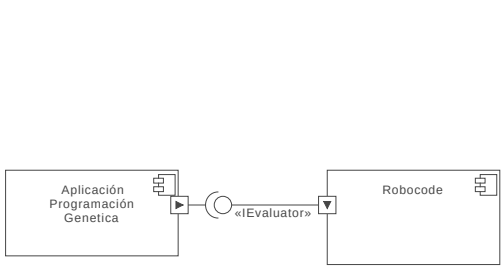
3.2.1. Software inicial

El software del que se disponía estaba organizado como una una aplicación monolítica. En ésta se encontraban integrados los operadores de la programación genética. Además, se encontraba adaptado para utilizar siempre los mismos operadores, sin apenas poder personalizar su configuración para cada ejecución.

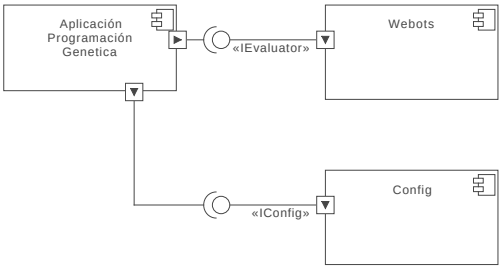
3.2.2. Método de diseño

El software empleado para realizar los procesos de búsqueda mediante evolución no se ha desarrollado desde cero. El software de partida[35] disponía de una estructura con responsabilidades distribuidas en componentes, aunque su estructura no disponía del suficiente grado de flexibilidad para todos los experimentos necesarios en este proyecto, dado que los componentes se encontraban demasiado acoplados. Por tanto, el método de diseño que se ha empleado principalmente ha adaptado una estructura de software existente a unos requisitos cambiantes y unas necesidades de estabilidad que se hicieron más estrictas con el paso del tiempo.

La estructura de componentes (Figuras 3.3.a y 3.3.b) no se ha visto afectada por los nuevos añadidos, aunque la estructura de los paquetes sí lo ha hecho considerablemente (Figuras 3.4.a y 3.4.b).

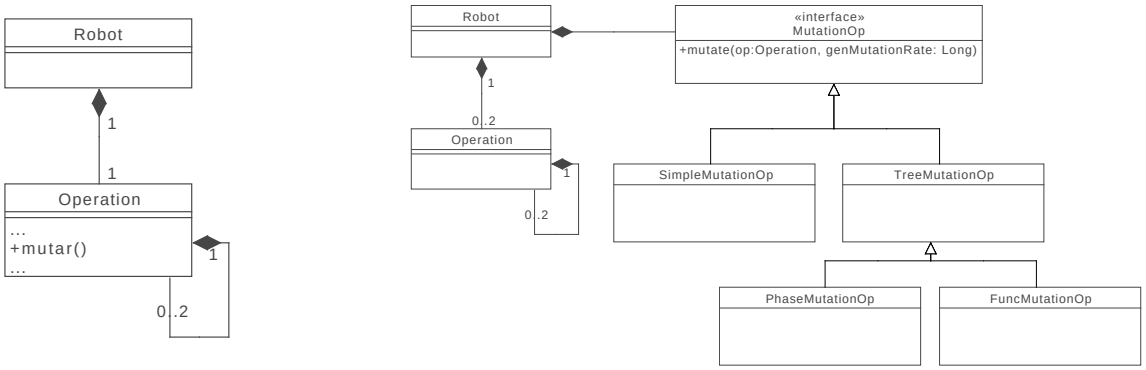


a) inicial



b) final

Figura 3.3: Diseño inicial vs final componentes



a) diseño inicial

b) diseño final

Figura 3.4: Diseño inicial vs final paquete referente a mutación

Por todos estos motivos, el método de diseño que se ha empleado para desarrollar la arquitectura se ha obtenido mediante:

- Refactorización de los componentes, a medida que el desarrollo ha ido añadiendo requisitos al conjunto inicial.
- Simplificación de los componentes abstraídos, y su división en múltiples clases atendiendo a las diferentes necesidades que el componente debía satisfacer.
- El principio de única responsabilidad que conlleva que los componentes satisfacen estrictamente una funcionalidad.

Como puede verse en la Figura 3.3.b, el diseño de componentes se apoya en los siguientes componentes:

- CID#1 - Aplicación de programación genética.
- CID#2 - Aplicación de evaluación (Webots).
- CID#3 - Librería para acceso a la configuración.

A continuación, se presenta una visión general de cada componente listado. La explicación detallada se facilita en el apéndice E.

- En primer lugar, el componente *aplicación de programación genética* se encarga de realizar tanto el cruce como la mutación a lo largo de las diferentes rondas que comprenden la evolución. Se encuentra estructurada en paquetes de acuerdo con la Figura 3.5.

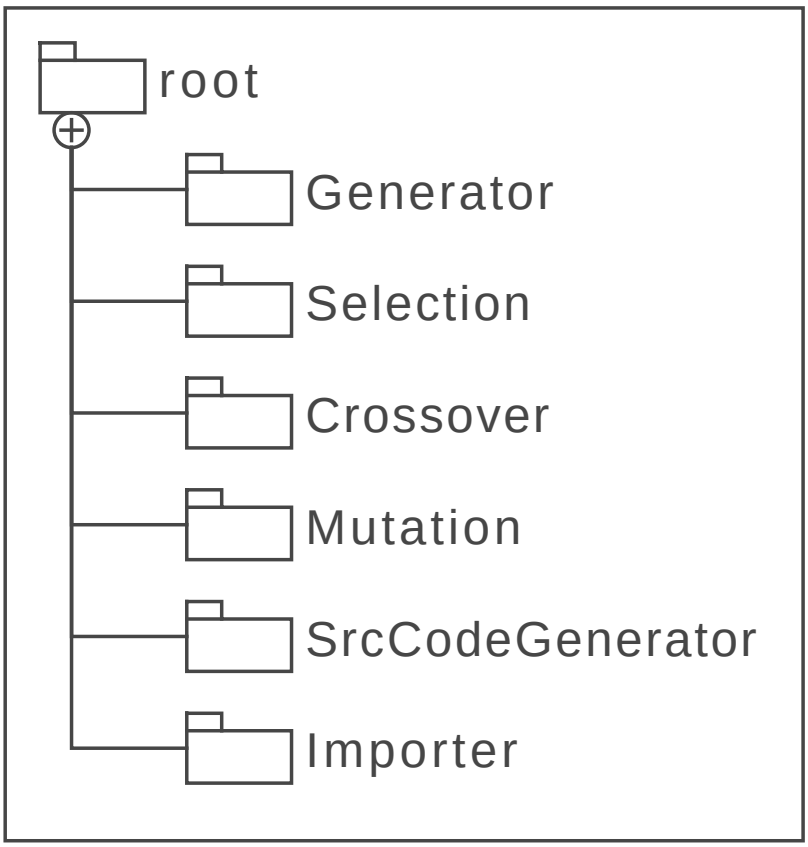


Figura 3.5: Estructura de paquetes del componente *Aplicación de programación genética*

  La especificación de los diferentes paquetes que integran este componente puede consultarse en el apéndice G.

- En segundo lugar, el componente *librería de acceso a Webots* permite utilizar la aplicación mencionada para realizar evaluaciones de los individuos y obtener su *fitness*. Su diseño de paquetes se encuentra reducido a un único paquete, dado que su conjunto de funciones a realizar es bastante reducido. La especificación de

este paquete puede consultarse en el apéndice G, así como la de su interfaz en el apéndice F.1.

- En tercer lugar, el componente *librería de acceso a parámetros* permite recuperar y gestionar los parámetros facilitados a la aplicación a través de la linea de comando y del fichero de configuración. Además, también resuelve qué valor debe utilizarse en caso de conflicto entre ambos. Este componente se apoya en la librería de *C++ boost*, que ofrece una funcionalidad ligeramente distinta a la requerida en esta aplicación. La especificación de este paquete puede accederse en el apéndice G, así como la correspondiente a su interfaz en el apéndice F.2.

3.3. Resultados

El sistema desarrollado dispone de un juego de pruebas asociado de tipo unitarias, de estabilidad y de memoria. Cada componente del sistema dispone de un juego de pruebas unitarias asociado que define el comportamiento de éste y que se ejecuta de forma automática. El desglose de las pruebas asociadas se muestra a continuación en la tabla 3.3:

Tabla 3.3: Desglose del volumen de pruebas por componentes

Componente	# Pruebas
Config	76
Crossover	53
Evaluator	3
Generator	74
Importer	84
Mutation	136
Selection	0
SrcCodeGenerator	22
Master	303
Total	751

Además de las pruebas unitarias, la aplicación pasa también pruebas de estabilidad. Estas pruebas consisten en ejecutar la aplicación con la propia configuración del experimento (sólo el numero de rondas a iterar se ve reducido), pero se omite la llamada al evaluador externo y se realiza una asignación de *fitness* aleatorio. El cambio se realiza para poder ejecutar la aplicación de forma acelerada y poder verificar así, que no se producen excepciones no controladas mediante la ejecución de la *suite* de pruebas.

Por último, la aplicación también pasa pruebas de memoria. Durante el desarrollo del proyecto se detectaron problemas de estabilidad relacionados con la gestión de la memoria, que impedían prolongar la ejecución de la aplicación más allá de una centena de rondas (generalmente, los fallos cortaban la ejecución antes). Estos problemas se debían a un complejo ciclo de vida de la memoria solicitada al sistema, que tenía errores complejos en la gestión (p.ej. al solicitar un clonado superficial de una lista de objetos en lugar de solicitar un clonado completo de éstos. En la primera, los objetos no se copian). Para resolver estos problemas, se han realizado pruebas de gestión de memoria con la aplicación *valgrind*. Desde entonces, estas pruebas han aportado un grado extra de estabilidad a la aplicación, así como han permitido tener bajo control las fugas existentes de memoria y minimizar su impacto en el sistema. Si se desea conocer el método de ejecución de las pruebas de memoria, puede consultarse el apéndice H.

3.4. Conclusiones

A partir del desarrollo realizado en este capítulo, se procede a formalizar las conclusiones alcanzadas:

- Desarrollar una librería propia de programación genética ofrece mayor nivel de versatilidad frente al uso de una librería existente.
- Mantener bajo control los fallos propios de su desarrollo y su continua adaptación requiere muchas pruebas y una buena organización.
- Pasar de un código demasiado acoplado sin pruebas a un código distribuido y con posibilidad de realizar pruebas automáticas requiere de una metodología de pruebas que permita evitar incluir errores en el código recién adaptado.

Capítulo 4

Modelo bípedo de locomoción

4.1. Introducción

En este capítulo se tratan los modelos bípedos de locomoción, que aglutinan todas las técnicas aplicadas para conseguir introducir un peatón con capacidad de locomoción bípeda (i.e., andar sobre dos piernas) en el entorno de simulación Webots, que es el objetivo del presente proyecto.

El conjunto de técnicas empleadas se dispone en las siguientes secciones: El proceso de creación del peatón virtual, el modelo ad-hoc 1 (*AH1*), el modelo ad-hoc 2 (*AH2*) y el modelo de ecuaciones diferenciales no lineales (*EDNL*).

Con el proceso de creación del peatón virtual se presentan cuatro aspectos: La motivación existente para su desarrollo, el modelo físico del peatón empleado, las fases de la creación en el entorno virtual y el proceso de calibrado de la fuerza en las articulaciones.

A continuación, dentro del modelo AH1 se muestran una descripción del modelo, la motivación de éste, cómo se aplicó la programación genética en este caso, qué experimentos se han realizado, así como las conclusiones a las que se ha llegado a partir de los resultados obtenidos.

Después del modelo AH1, se pasa al modelo AH2. En esta parte también se detallan los mismos contenidos del modelo AH1 referidos a este modelo.

Terminado el modelo AH2, se muestra el modelo de ecuaciones diferenciales no lineales utilizado. Esta parte de la sección muestra una estructura muy similar a los modelos AH1 y AH2, aunque la parte de aplicación de programación genética (*PG*) es más amplia, puesto que la PG se aplicó de diversos modos. Los resultados de este modelo, por tanto, se muestran asociados a cada una de las formas en las que se aplicó la PG. Esto es, en cada sección de aplicación de *PG* se muestran los experimentos realizados y los resultados obtenidos.

Por último, se indican las conclusiones alcanzadas de este capítulo en la última sección.

4.2. Proceso de creación del peatón virtual

El proceso de creación del peatón virtual resulta de vital importancia para comprender la dificultad del proyecto, puesto que se pretende disponer de un peatón que camine sobre dos piernas en el entorno de simulación *Webots*. Esto viene motivado, porque *Webots* dispone de un entorno urbano para probar un vehículo autónomo, que incluye calles, edificios, vegetación, pasos de cebra, pero no incluye peatones. Si se quisiera probar algoritmos de control de vehículos autónomos, no podría realizarse una simulación completa.

Sin embargo, *Webots,* en la versión 6.3.4, solamente dispone de dos modelos de robot bípedo. El primero de ellos, *NAO*[46], mide 58 cm y pesa 4,3 kg. El segundo, *HOAP*[47], es algo más pequeño (mide 48 cm), pero más pesado (6 kg), por lo que ninguno es apto para representar a un peatón común.

Esto llevó a tener que modelar un peatón en el contexto de este proyecto.

4.2.1. Modelo de peatón adoptado

A continuación se presenta el modelo de peatón adoptado para este proyecto. El modelo de peatón se corresponde con la distribución de la masa de éste a lo largo del volumen que ocupa cada uno de sus miembros. Esta distribución cobra mucha relevancia, puesto que el motor físico actuará sobre el peatón de distinto modo si el peso se encuentra distribuido de otra forma, pudiendo incluso impedir el movimiento del peatón.

Este modelo se ha inspirado entre varios disponibles. Dempster y Gaughran[44] plantean una distribución del peso en porcentaje por segmentos que puede verse en la Tabla 4.1. Harless[26] dispone una distribución del peso de forma muy similar que puede verse en la Tabla 4.2. Drills, Contini y Bluestein[45] establecen que la distribución del peso de una persona se asemeja más a la mostrada en la Tabla 4.3.

Tabla 4.1: Distribución del peso en los segmentos del cuerpo en porcentaje de acuerdo con las mediciones llevadas a cabo por Dempster y Gaughran.

Segmento	% Peso
Cabeza	7,92 %
Tronco	43,37 %
Parte superior del brazo	2,64 %
Parte inferior del brazo y mano	2,14 %
Parte superior de la pierna	10,01 %
Parte inferior de la pierna	4,61 %
Pie	1,43 %

Tabla 4.2: Distribución del peso en los segmentos del cuerpo en porcentaje de acuerdo con las mediciones llevadas a cabo por Harless.

Segmento	% Peso
Cabeza	7,04 %
Tronco	49,29 %
Parte superior del brazo	2,76 %
Parte inferior del brazo y mano	2,32 %
Parte superior de la pierna	10,88 %
Parte inferior de la pierna	4,42 %
Pie	1,45 %

Tabla 4.3: Distribución del peso en los segmentos del cuerpo en porcentaje de acuerdo con las mediciones llevadas a cabo por Drills, Contini y Bluestein.

Segmento	% Peso
Cabeza	7,12 %
Tronco	46,28 %
Parte superior del brazo	3,23 %
Parte inferior del brazo y mano	2,60 %
Parte superior de la pierna	11,20 %
Parte inferior de la pierna	4,37 %
Pie	1,82 %

La distribución utilizada en este proyecto se corresponde con la que se muestra en la Figura 4.1, que se encuentra en sintonía con la dispuesta tanto en [26] como [44], así como en [45]. La distribución del peso de una persona puede variar con respecto a la distribución general. Por lo tanto, basándose en la distribución porcentual del peso según Harless [26], se ha extrapolado esa distribución para una persona de $1,90m$ y $91,23kg$ de peso de la forma que se ve en la Figura 4.1.

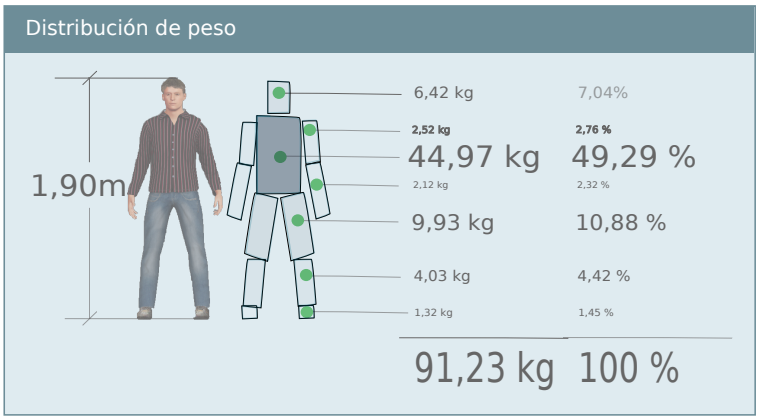


Figura 4.1: Distribución del peso corporal en el peatón virtual

En la Figura 4.1 puede apreciarse como el peso se concentra, principalmente, en el torso y los miembros inferiores . Esto conlleva que los movimientos de los brazos no puedan generar un desequilibrio por sí solos, pero ofrecen un contrapeso para mantener el equilibrio sobre dos piernas con mayor facilidad.

A continuación se presenta el proceso seguido para implementarlo dentro de *Webots.*

4.2.2. Fases de creación en el entorno virtual

Una vez se ha mostrado el modelo de peatón que se ha implementado en *Webots*, se presentan las diferentes fases que se han seguido para introducir este modelo en el simulador.

En este punto conviene aclarar algunos conceptos, que se utilizarán a continuación:

- *Malla poligonal* o *malla*: se trata de una superficie creada mediante un método tridimensional generado por sistemas de vértices posicionados en un espacio virtual con datos de coordenadas propios [48].

Makehuman: Malla inicial

En una primera fase se realiza una generación del modelo gráfico de un ser humano, es decir, la malla poligonal del peatón. Ésta se llevó a cabo utilizando el software de código abierto *MakeHuman*[36]. Este software permite generar modelos humanos con cierto nivel de personalización. La Figura 4.2 muestra una captura de la aplicación en su estado actual.

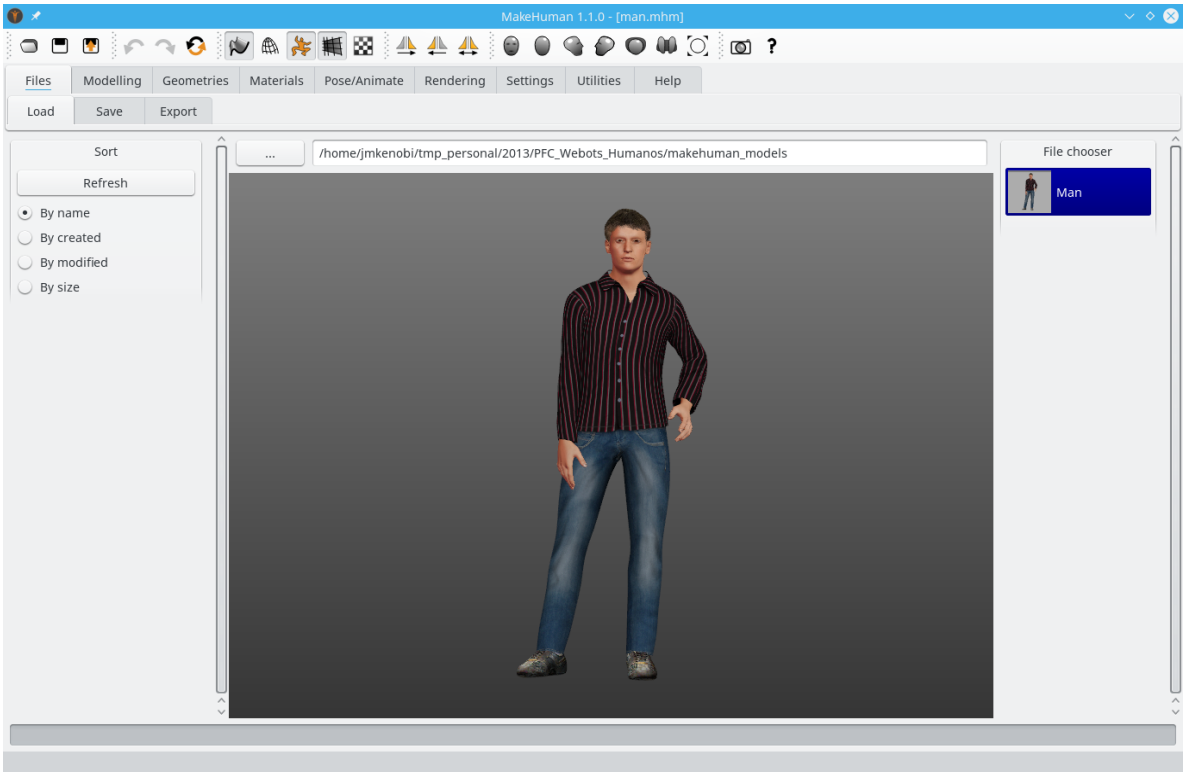


Figura 4.2: Creación de modelo usando MakeHuman

Sin embargo, este software no permite generar un modelo que se pueda importar en *Webots* directamente, por lo que es necesario realizar 2 conversiones de formato hasta poder importar la malla en *Webots* (véase Figura 4.3). Estas conversiones tendrán consecuencias que se explicarán más adelante.

$$\text{MakeHuman} \xrightarrow[\text{Blend}]{} \text{Blender} \xrightarrow[\text{X3D}]{} \text{Meshlab} \xrightarrow[\text{VRML}]{} \text{Webots}$$

Figura 4.3: Conversiones de formato para la importación en Webots

Blender: Segmentación y conversión de la malla

A continuación, el modelo se importa en *Blender*[37]. Inicialmente se utilizó para realizar una conversión al formato *X3D* (*Blender* no permitía exportar directamente a *VRLM* 1.0). Sin embargo, una primera ejecución del proceso de importación (Figura 4.3) de la malla

en *Webots* reveló que la malla se importaba en *Webots* de forma indivisible y no manipulable (i.e., no se puede cambiar de postura), a pesar de que la malla disponía de una definición de articulaciones, que permitían mover los miembros del peatón y modificar la malla con naturalidad. La exportación al formato VRML 1.0[39] (éste es el formato con el que Webots permite la importación de mallas) elimina estas definiciones.

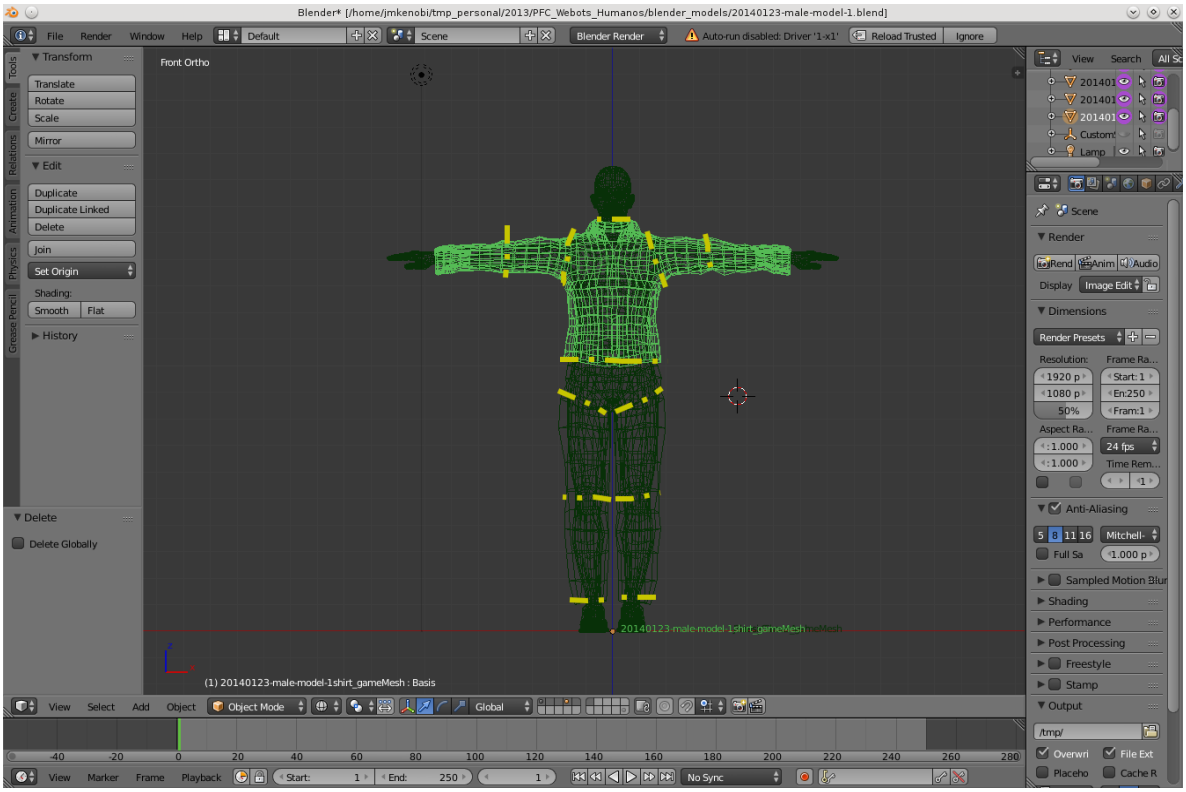


Figura 4.4: Malla de peatón con partes segmentadas en la aplicación Blender previo a la separación de las partes de la malla.

En consecuencia, se volvió necesario realizar una descomposición en partes del modelo humano e importar por separado cada una de las mallas requeridas en *Webots*. Y para realizar esta segmentación del modelo en subpartes se utilizó *Blender*. La Figura 4.4 muestra una imagen de las diferentes partes en *Blender*. De este modo puede conseguirse la composición del modelo humano ya manipulable en el entorno *Webots*.

Meshlab: Conversión a formato Webots

Blender no ofrece exportación en ningún formato desde el que *Webots* pueda importar directamente el modelo, por lo que es necesario realizar nuevamente otra fase de transformación intermedia. En esta conversión se emplea el programa *Meshlab*[38], que puede apreciarse en la Figura 4.5. Dicha figura muestra el segmento correspondiente a la parte superior del cuerpo, importado en *Meshlab*, para exportarla al formato que admite Webots (VRML[39]).

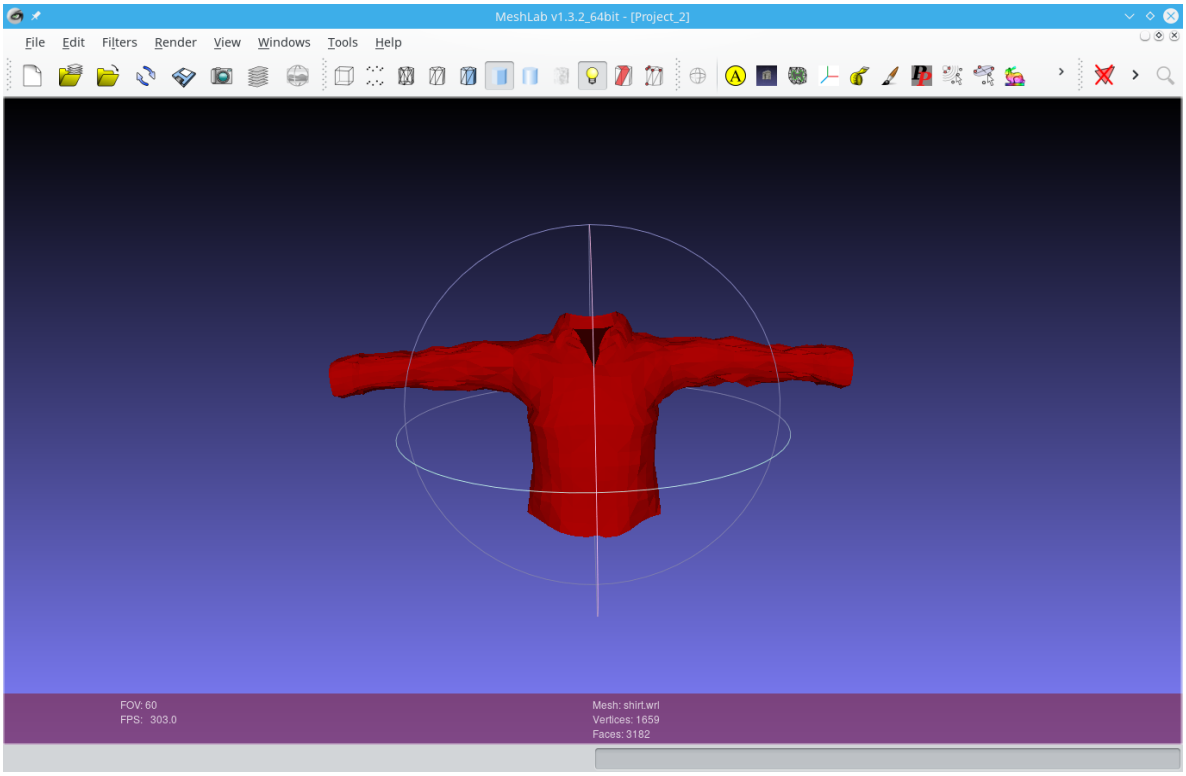


Figura 4.5: Conversión mediante Meshlab

Webots: Composición del modelo

La composición final del modelo de peatón se realiza ya en el entorno de simulación *Webots*. Ésta se realiza mediante la importación de los diferentes segmentos producidos en *Blender* y su posterior conversión vía *Meshlab,* los cuales se ordenan de acuerdo con su posición en el cuerpo humano. Con esta configuración se crea el modelo visual dentro

de *Webots*, el cual determinará la apariencia del peatón. Esta configuración se mantuvo inalterada hasta la finalización del proyecto.

Sin embargo, el modelo creado hasta el momento dispone de un modelo visual, pero no interactúa con el motor físico, puesto que carece de modelo físico.

Inicialmente, se construyó un modelo físico que constaba de formas similares a los miembros del cuerpo humano. Este modelo constaba de asociaciones entre complejos volúmenes (i.e., tenían la forma del miembro modelado) y sus masas, que permitían al modelo físico asemejarse a la forma de los miembros del peatón modelado. Sin embargo, el procesamiento de estas asociaciones volumen-masa por parte del simulador no se ajustaba a las expectativas y se producían errores en la simulación. El motor físico realizaba el cálculo de fuerzas sobre las asociaciones, y como consecuencia, el modelo del peatón se separaba del suelo virtual con una velocidad constante. En la Figura 4.6 puede verse el efecto producido en la simulación.

Figura 4.6: Consecuencias modelo físico complejo

Posteriormente, se optó por realizar una simplificación de este modelo como solución. Ésta transformaba los volúmenes complejos de las asociaciones volumen-masa del modelo físico del peatón en volúmenes ortoédricos, de mayor simplicidad para calcular las fuerzas. Las medidas de los volúmenes de este segundo modelo se encuentran adaptadas para representar lo más fielmente el modelo original. La Figura 4.7 muestra el contraste entre la representación física inicial (izquierda) y final (derecha).

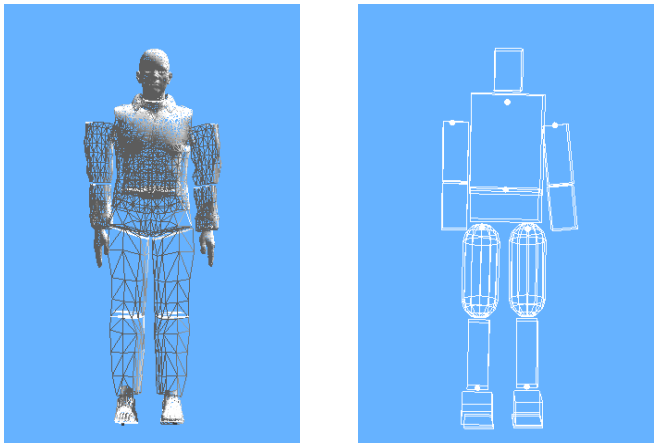


Figura 4.7: Comparación de detalle de los modelos inicial y final

4.2.3. Calibrado de fuerza de articulaciones

Con la realización de las fases explicadas en la sección anterior se introdujo un modelo de peatón en el entorno de simulación *Webots*. Sin embargo, el movimiento que ofrecía dicho modelo no se ajustaba a la realidad. Por lo que fue necesario calibrar la fuerza en cada articulación para permitir un movimiento más similar al movimiento natural del cuerpo humano. La fuerza que permite realizar cada articulación es distinta y no se partía de un modelo conocido con anterioridad, por lo que su calibrado tuvo que realizarse mediante prueba y error. El objetivo era minimizar la diferencia entre los efectos causados por una articulación real y su versión simulada, para dotar de mayor realismo a la simulación.

Este proceso de calibrado no fue una tarea sencilla y requirió construir un entorno de simulación adaptado para poder realizar su calibrado de forma efectiva. La Figura 4.8 muestra el peatón en dicho entorno y los diferentes tipos de calibrado realizados para poder determinar de forma efectiva las fuerzas necesarias en cada una de las articulaciones.

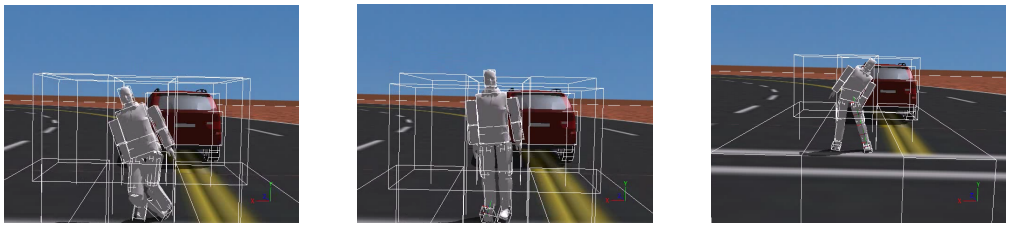


Figura 4.8: Calibrado de articulaciones inferiores. En la izquierda, calibrado de cadera, rodilla y tobillo en el eje X. En el centro, calibrado de tobillo eje Z. En la derecha, calibrado de cadera eje Z.

En primer lugar, se calibraron las articulaciones de la cadera, rodilla y tobillo en el eje X. La torsión alrededor de este eje es la responsable de que el peatón pudiese sentarse en los tobillos de forma perpendicular al suelo (también participa del movimiento de avance del peatón). Para ello se incluyó al peatón entre volúmenes ortoédricos mucho más pesados, que sólo le permitían realizar un movimiento arriba y abajo (ponerse de pie - sentarse). Por lo que se fue incrementando la fuerza disponible en las articulaciones (Cadera eje X, rodilla y tobillo eje X) en ambas piernas hasta permitir que el peatón pudiese realizar este movimiento. Una imagen de este proceso puede verse en la Figura 4.8 (izquierda).

En segundo lugar, se calibró la fuerza del tobillo lateralmente (eje Z). Esta articulación debe permitir que el peatón se mantenga equilibrado y no se ladee ni se caiga lateralmente, con ayuda de la articulación cadera eje Z. Sin embargo, esta última se calibró aparte. Este calibrado también se realizó en el mismo entorno que el anterior, pero ofrecía algo más de libertad para desplazar el cuerpo lateralmente. El calibrado se dio por terminado una vez que accionar los tobillos en una dirección desplazaba todo el cuerpo lateralmente. Esto puede verse en la Figura 4.8 (centro).

En último lugar, se calibró la fuerza de la cadera eje Z. Ésta es responsable de inclinar lateralmente el tronco del cuerpo. Para realizar este calibrado se incluyó al peatón en un entorno similar al anterior, pero que le permitía mover el tronco a un lado y a otro, y que en todo momento mantenía al peatón de pie. En cuanto se pudo mover el tronco del peatón de un lado a otro de forma controlada, se consideró finalizado su calibrado. La Figura 4.8 (derecha) ilustra este último calibrado.

Las articulaciones que fue necesario calibrar se encuentran listadas a continuación:

- Cadera eje X
- Cadera eje Z
- Rodilla
- Tobillo eje X
- Tobillo eje Z

Inicialmente, no se consideró calibrar la fuerza en las articulaciones de los brazos, puesto que éstas no influían de forma determinante para el desplazamiento del robot. Tampoco se calibró la fuerza de la cadera eje Y, puesto que el movimiento mostrado por esta articulación se asemejaba a la realidad.

4.3. Modelos bípedos de locomoción empleados

A continuación se presentan los diferentes enfoques que se han tenido en cuenta para desarrollar un peatón que tuviese capacidad de caminar. Con este fin se han utilizado dos enfoques distintos:

1. Enfoque *ad hoc*. El sistema de programación genética genera el modelo de forma intrínseca.
2. Enfoque basado en el modelo de ecuaciones diferenciales [24]. El control de cada una de las articulaciones viene determinado por una ecuación diferencial. En este caso, el sistema evolutivo ayudará a determinar el ajuste de las ecuaciones en el modelo dado.

4.3.1. Aproximación mediante modelos *ad hoc*

Dentro de este enfoque se explican las diferentes técnicas que se han empleado para obtener un modelo de locomoción bípedo.

Aproximación 1 - Modelo 1

El primer modelo que se propuso busca modelar el movimiento del peatón como una secuencia de acciones coordinadas para cada una de las articulaciones. Esto es, indicar los ángulos de rotación de una articulación, p. ej. la cadera en el eje de avance (eje X), durante una serie de periodos de tiempo no necesariamente iguales (i.e., [40º, 100ms],[45º, 150ms]...). Mover las articulaciones siguiendo estas indicaciones puede producir un movimiento continuo que permita caminar.

La codificación de este modelo en la aplicación de programación genética consiste en crear un árbol para cada secuencia de instrucciones de un periodo de tiempo. Cada nodo de dicho árbol es una instrucción para un actuador, en el que el orden no importa. Si se tienen seis articulaciones, el árbol resultante dispondrá de seis nodos como mucho, pues se puede no querer modificar el estado de una articulación.

No obstante, esta codificación necesita ciertas adaptaciones para poder utilizarlo dentro de Webots. La coordinación entre actuadores, sensores y el controlador del peatón virtual se realiza utilizando una instrucción de sincronización, de modo que todas las acciones especificadas sobre los actuadores se realizan al ejecutar esta instrucción. Esta instrucción recibe el nombre de *step.* Las instrucciones dadas a los actuadores desde la última ejecución de sincronización y la siguiente se considera un periodo.

Los árboles generados para cada periodo, o *subárboles de periodo*, se van a integrar en un único árbol. La integración se realiza, en primer lugar, creando un árbol con las instrucciones *step,* o *árbol de step,* cuyas instrucciones delimitan cada periodo (la duración de éste). En segundo lugar se cuelgan de los nodos hoja de este árbol cada uno de los subárboles de periodo creados anteriormente. De este modo, un recorrido en-orden de los nodos del árbol produce la secuencia de instrucciones del movimiento del peatón ordenada por periodo. La Figura 4.9 ilustra esta descripción.

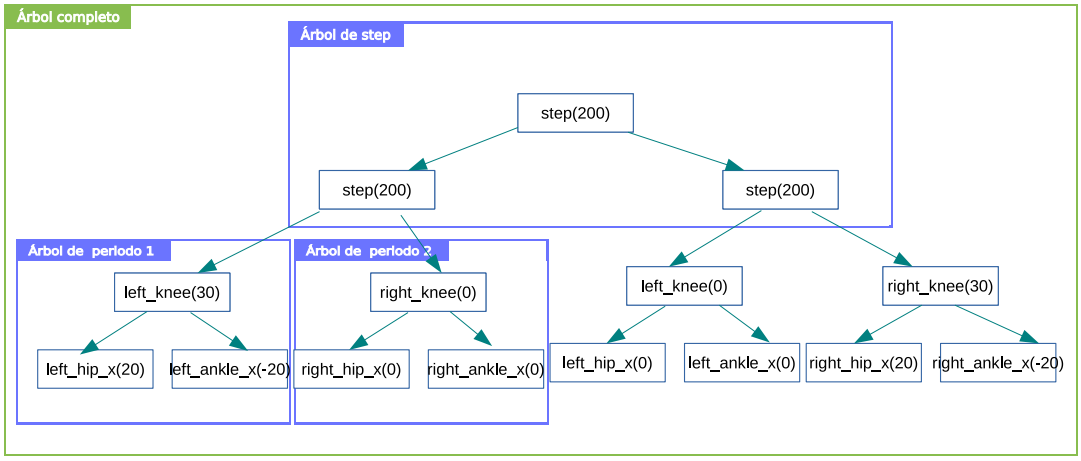


Figura 4.9: Representación en árbol del individuo, con nodos articulación

La aplicación de los operadores genéticos se realiza a nivel de nodo del árbol completo. Sobre esta estructura puede producir nuevos periodos o una reducción de estos, provo-

cando en este caso que se repitan instrucciones para la misma articulación. Si se tiene en cuenta que solamente se ejecuta una instrucción por actuador y que la ejecutada es la última en caso de duplicados, se observa que los árboles pueden disponer de código inútil. Para asegurar la efectividad de los operadores genéticos se desarrolló un componente que analizaba dichos árboles y los simplificaba, eliminando los nodos repetidos de un subárbol de periodo. La evaluación de los individuos de este modelo se realiza teniendo en cuenta la distancia recorrida. En una primera fase utilizando este modelo, no se realizó ningún control acerca de la duración del bucle de acciones para caminar. En una segunda fase, esta duración se limitó a dos segundos.

Utilizando este modelo se han realizado 17 experimentos (ver Figura 4.10). En ellos se han realizado múltiples variaciones tanto de porcentajes de cruce, de mutación, de elitismo y tamaño de torneos para selección. Los resultados son bastantes similares para todos ellos, mostrando que este modelo ofrecía una media de 2,36m de distancia recorrida y un tiempo en el que el peatón se encontraba de pie de 4,64s. Además, resultan visibles las dos fases diferenciadas en la ejecución de estos experimentos. Estos primeros experimentos tenían una duración media de 6,47 días y un máximo de 17 días de duración. Por lo que se tardó bastante en obtener el feedback en algunos casos. En una primera fase, el modelo, que no tenía ninguna limitación en relación al cuerpo del bucle, mostraba una distancia recorrida media de 3,01m y una duración de pie de 5,95s. En la segunda fase, tanto la distancia recorrida como el tiempo de pie se ven reducidos, al introducirse el control de duración de bucle (1,98m de distancia, 3,86s de duración de pie).

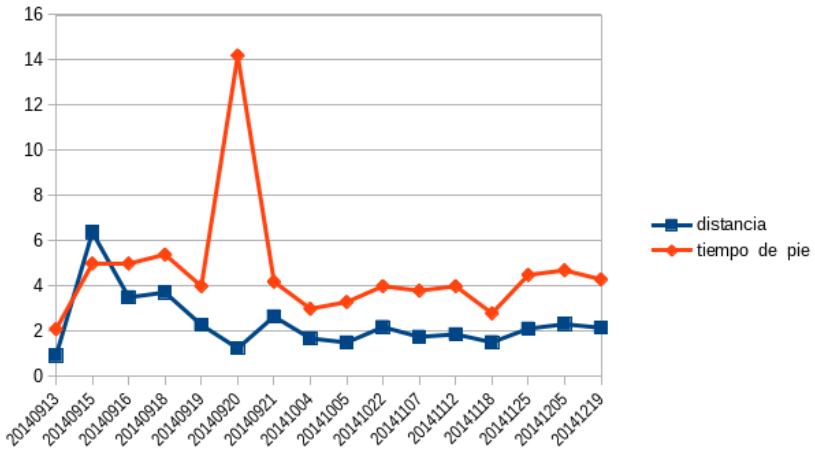


Figura 4.10: Experimentos realizados con este modelo

Esta reducción en la distancia recorrida y en el tiempo de pie del peatón se puede explicar viendo los mejores métodos de caminar desarrollados mediante programación genética antes y después de introducir la limitación de los dos segundos. Antes de introducir dicha limitación, el proceso evolutivo llevaba a los individuos a tener cuerpos de bucle cada vez más largos. Un mayor tiempo de evolución producía un método de caminar que duraba más tiempo caminando. Sin embargo, al introducir la limitación de los dos segundos, se observó que los modelos generados se limitaban a producir desplazamiento hacia adelante dando pequeños saltos, siendo además más inestable. En el mejor de los casos, no superaban las dos iteraciones del bucle (2s de duración/iteración). Al finalizar esta parte de los experimentos se decide implementar pruebas automáticas en el algoritmo de programación genética, con el fin de mejorar la calidad de la aplicación, mediante la validación del comportamiento de la aplicación de programación genética y la eliminación los defectos existentes en el código fuente en ese momento. Se necesitaba comprobar que la aplicación no impedía el correcto funcionamiento del proceso evolutivo.

En conclusión, la aplicación de este modelo produjo comportamientos que permitían la locomoción bípeda hacia adelante, aunque ésta carecía de estabilidad y estaba muy limitada en el tiempo. Además, no producía una secuencia repetible que permitiese caminar al peatón de forma continua. Una vez se encontraron disponibles las pruebas automáticas, se subsanaron los errores detectados. Y una posterior ejecución determinó que se necesitaban más cambios, por lo que se decidió cambiar ligeramente el modelo, por si este fuese el problema.

Aproximación 2 - Modelo 2

El segundo modelo que se propuso busca también modelar el movimiento del peatón como una secuencia de acciones coordinadas para cada una de las articulaciones. Sin embargo, estas acciones se agrupan temporalmente en periodos, los cuales se denominan *poses*.

La codificación de este modelo en la aplicación de programación genética se realizó de forma muy similar al modelo anterior. Cada nodo en el *subárbol de periodo* es una instrucción para un actuador y tampoco importa el orden en el que se indique. Además, un recorrido en-orden del árbol debe producir la misma secuencia de instrucciones del movimiento del peatón que en el modelo anterior. La diferencia radica en el modo en que los operadores genéticos tratan el árbol completo. El operador de cruce, a diferencia del

modelo anterior, solamente selecciona poses para intercambiarlas con otro individuo y no nodos individuales cualquiera (véase Figura 4.11 y Figura 4.12).

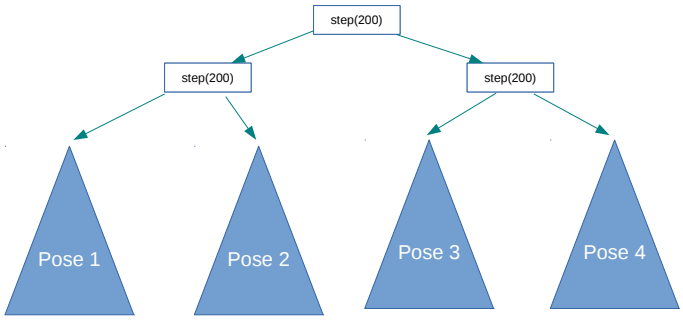


Figura 4.11: Representación en árbol del individuo, con nodos articulación y poses

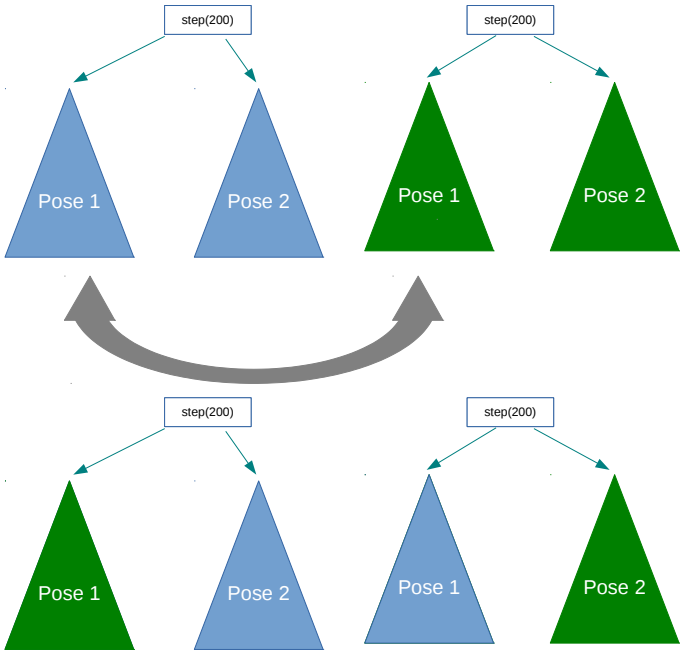


Figura 4.12: Modo de operación de cruce utilizando poses

El único operador que puede alterar las poses en este modelo es el operador de mutación. Por lo que en este modelo el número de poses que puede tener un comportamiento de caminar (o individuo de la población) se encuentra limitado en el momento de la generación. Al igual que en el modelo anterior, también podía encontrarse código inútil en la pose debido a repeticiones de instrucciones para el mismo actuador, por lo que también se aplicó el componente de simplificación para eliminar los nodos repetidos. Puesto que el objetivo de este proyecto es generar un método que permita al peatón caminar de forma estable, se necesitaba generar un bucle que pudiera repetirse las veces que fuese necesario. Por tanto, se mantuvo incluida la limitación de la duración del bucle de dos segundos.

Con este modelo se han hecho 4 experimentos, pues se trata de un modelo muy similar al anterior. Uno de ellos se encuentra dañado, y por tanto, se deja fuera del análisis comparativo realizado. En este caso también se han realizado múltiples variaciones tanto de porcentajes de cruce, de mutación y de elitismo, como en el tamaño de torneos para realizar la selección.

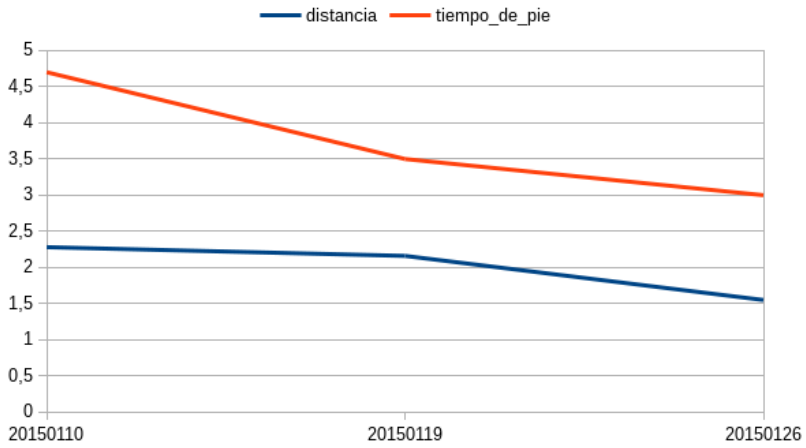


Figura 4.13: Experimentos realizados con este modelo

Los experimentos realizados muestran una continuidad con respecto al modelo anterior. Estos no ofrecen una mejora significativa respecto a la distancia recorrida del peatón, ni respecto al tiempo que éste se puede mantener caminando. Además, este modelo revela que no puede generar un controlador con un bucle que permita caminar al peatón virtual.

Aproximación 3 - Modelo 1 con entorno especial

En tercer lugar, se mantuvo el enfoque original, pero esta vez se modifica el entorno virtual para que incluya limitaciones simuladas. Estas limitaciones irían paulatinamente permitiendo un mayor grado de libertad en los movimientos del peatón en diferentes fases. A medida que los individuos alcanzasen una mayor distancia recorrida, irían atravesando cada una de las distintas fases existentes en el entorno simulado. La Figura 4.14 muestra el entorno generado para este fin.

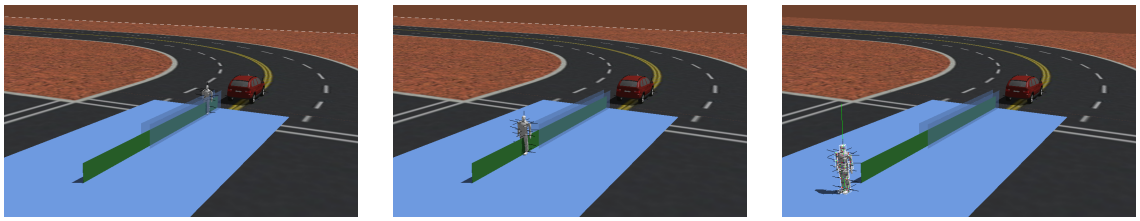


Figura 4.14: Modelo 1 con entorno de simulación limitado

En una primera fase, el movimiento del peatón se encuentra muy restringido y sólo le permite realizar movimientos en aquellas articulaciones cuyos ejes se encuentren perpendiculares al eje de avance del peatón.

En una segunda fase, el movimiento del peatón se encuentra parcialmente limitado, pero ya se le permite mover otras articulaciones, aunque sus ejes de rotación no se encuentren perpendiculares al eje de avance del peatón.

En una última fase, se le retiran todas las restricciones al peatón y se le deja moverse con libertad. Llegados a este punto, se supone que el peatón caminaría de forma estable.

Sin embargo, la configuración de dicho entorno con la aplicación de programación genética no se pudo evaluar. Entre las limitaciones del motor físico de Webots (versión personalizada de ODE) se encuentra que el peatón virtual no puede interaccionar con un objeto virtual compuesto por varias partes que tienen modelo físico distribuido (i.e., nodos con pesos que forman parte del mismo objeto).

Una vez se vio que los resultados conseguidos con los modelos *ad-hoc* no iban a proporcionar el controlador deseado, se pasó a utilizar el modelo de ecuaciones diferenciales.

4.3.2. Aproximación mediante modelo de ecuaciones diferenciales

En este enfoque el movimiento de cada articulación para el intervalo siguiente se define mediante una ecuación, que tiene en cuenta el ángulo anterior de la articulación para producir el cambio necesario en la articulación. Este tipo de ecuaciones, que tienen en cuenta el valor inmediatamente anterior, se denominan ecuaciones diferenciales. Además, se utiliza una ecuación diferencial a modo de generador que produce valores rítmicos y casi repetitivos. Al emplear esta función, las diferentes ecuaciones del modelo se sincronizan y permiten un movimiento de articulaciones armónico, que puede permitir caminar al peatón. Este modelo se encuentra descrito en detalle en [55].

Los valores de la ecuación se calculan utilizando un método numérico. Aunque existen diferentes métodos (método iterativo de Euler, método iterativo refinado de Euler, los métodos de tipo Runge-Kutta) para calcular las ecuaciones diferenciales ordinarias (*EDO*), se ha seleccionado el método iterativo de Euler para calcular las aproximaciones a los valores reales de la ecuación diferencial, porque es el método más sencillo y porque esperamos que el algoritmo de programación genética pueda compensar los errores de cálculo para producir un controlador que permita al peatón virtual caminar. Todo esto a pesar de que el método empleado pueda no ser el más estable para calcular el valor de la ecuación diferencial en un punto dado.

En este modelo se desconoce la fórmula exacta del movimiento del peatón necesaria para cada articulación, sin embargo, se dispone de ecuaciones diferenciales no lineales parametrizables que podrían adaptarse para ello. Y en este proceso de adaptación se va a utilizar la programación genética para hallar los pesos o funciones modificadoras de la ecuación que permitan conseguir este controlador. La parametrización de las fórmulas facilitada en la publicación no es de utilidad para hacer caminar este peatón debido a las dimensiones de éste, mucho mayores que los robots indicados en ésta.

La codificación de este modelo consiste en crear un árbol de operadores ($+, -, \times, \div$) y funciones ($seno, coseno, exponencial, sigmoide\ y\ step$) para cada articulación que se desee modelar. No obstante, no se modelan todas las articulaciones. Solamente se crean modelos para las articulaciones de la cadera en sus tres ejes de rotación y la rodilla. En el caso del tobillo se utiliza la función inversa del eje en cuestión para determinar su ángulo en cada intervalo. Esto es, para el eje de avance (eje X) se utiliza la inversa de la suma de las ecuaciones de la cadera y de la rodilla. Tampoco se modelan las articulaciones simétricas, dado que el movimiento de los dos lados del cuerpo se encuentra acompasado con una diferencia de π rad, es decir, medio ciclo.

En una primera aproximación se aplicó la programación genética sobre el modelo cuatro articulaciones/partes a evolucionar. Aplicar este modelo consistió en evolucionar cuatro ecuaciones diferenciales para cada individuo. El primer modo de aplicación de la PG sobre este modelo puede verse en la Figura 4.15.

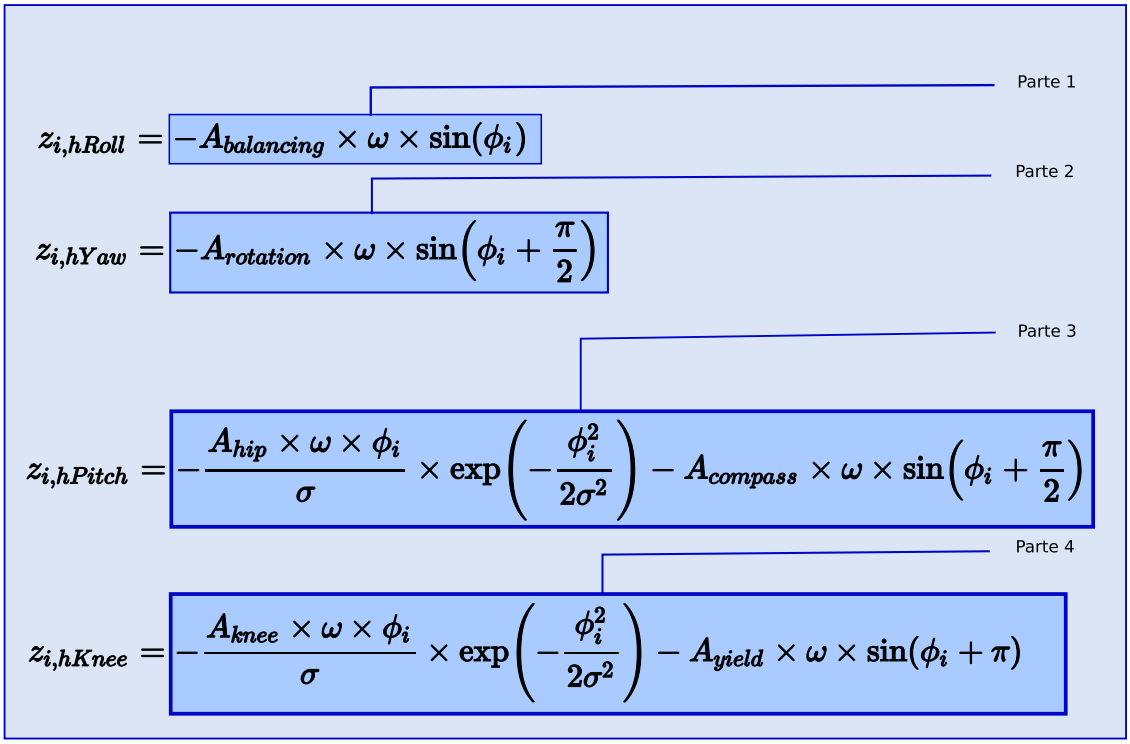


Figura 4.15: Primer modo de aplicar la programación genética. Se evolucionan las cuatro ecuaciones completas.

En una segunda aproximación se estipularon doce partes del modelo que se evolucionarían de forma independiente. Este modelo consiste en evolucionar por separado algunas partes de las ecuaciones diferenciales. Este modo de aplicar la programación genética puede verse en la Figura 4.16. Con este modo se buscaba mantener más la estructura de las ecuaciones originales.

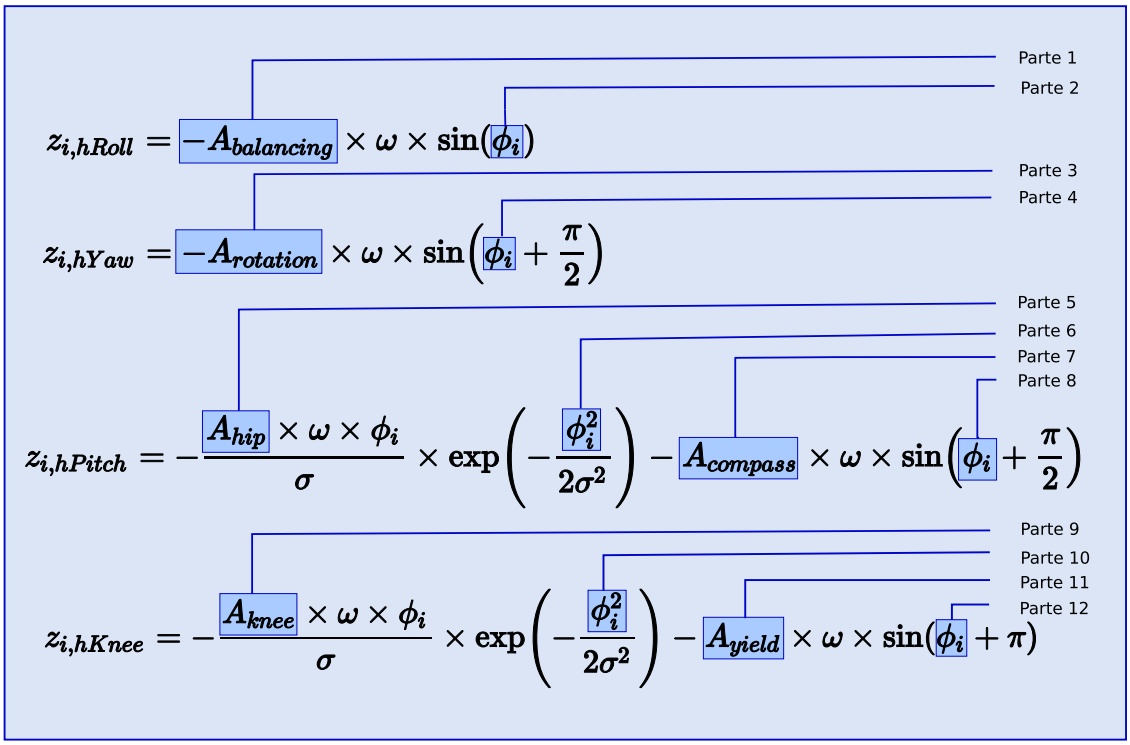


Figura 4.16: Segundo modo de aplicar la programación genética. Se evolucionan doce partes seleccionadas para mantener la estructura.

En una tercera aproximación se indicaron diez partes del modelo que se podían evolucionar independientemente. En este modelo también se busca evolucionar algunas partes de las ecuaciones por separado.

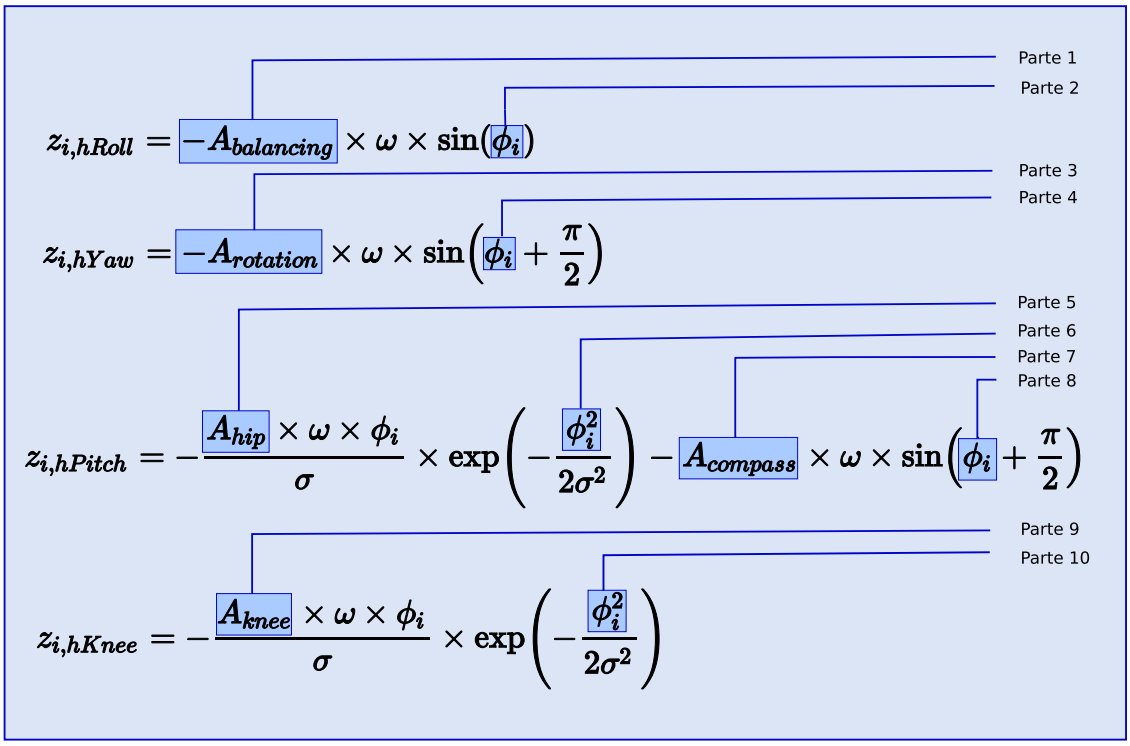


Figura 4.17: Tercer modo de aplicar la programación genética. Se evolucionan diez partes seleccionadas. Se usa un modelo ligeramente distinto.

La Figura 4.17 ilustra este modo de aplicar la programación genética. En este caso el modo de caminar es distinto, puesto que la rodilla carece de componente de impulso o *yield* (segunda parte de la ecuación) en la ecuación. Al tratarse de un modo de caminar distinto, solamente necesitan evolucionarse diez partes en lugar de las doce del modelo anterior. Los parámetros concretos de estos experimentos pueden consultarse en el Apéndice D.

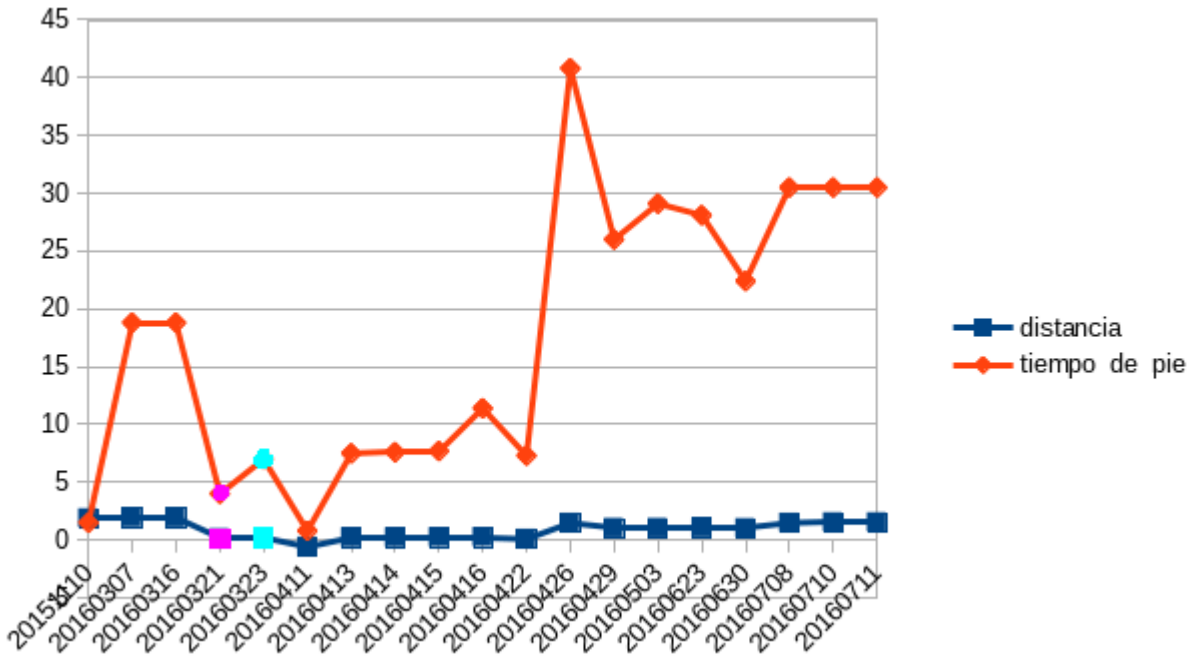


Figura 4.18: Primeros resultados de evolución utilizando el modelo EDNL

Los resultados de estos experimentos con los tres modelos indicados (ver Figura 4.18) muestran que el cambio de modelo no produjo necesariamente el controlador deseado. Tal y como se muestra en la Figura 4.18 los experimentos con este modelo hasta este punto no consiguen mejorar los resultados obtenidos por los modelos adhoc y siguen teniendo una duración prolongada en el tiempo que impide medir los efectos de cambios en parámetros estructurales del proceso evolutivo. Además, se muestra como los resultados no se ven afectados por el cambio de modelo en la aplicación de la programación genética (de cuatro[20160316] se pasa a doce y diez [20160321-violeta, 20160323-azul claro], respectivamente. Se vuelve a utilizar de nuevo cuatro partes a evolucionar en el experimento [20160411]). No obstante, se muestra un aumento del tiempo que se mantiene de pie, pues se modificó la función de evaluación para que no cancelara la simulación, aunque el controlador girase demasiado al peatón virtual. Además, para los últimos tres experimentos se utiliza un modo de búsqueda basado solamente en mutación mediante el empleo de seis tipos de ésta (*Grow*, *Shrink*, *AllNodes*, *OneNode*, *Swap*, *Gaussian*). Aunque parece mostrar una leve mejoría en la gráfica, no resulta significativa.

4.3.3. Aproximación mediante modelo de ecuaciones diferenciales con corrección de parámetros

En los experimentos realizados hasta este punto se han utilizado los parámetros dados en [54] y variaciones sobre éstos. Sin embargo, se constata que debe existir algún problema que impide alcanzar el objetivo. Por lo que se decide cambiar el dominio en el que se aplica la programación genética a hacer extracción de fórmulas a partir de tuplas de tipo [x,y].

La adaptación a este dominio se realiza con un mínimo coste de desarrollo de software. Entre las adaptaciones que se realizan, se incluye la posibilidad de determinar un porcentaje de aplicación de cada tipo de mutación, es decir, se puede especificar que del total de individuos que se van a mutar, se desea aplicar la mutación de tipo gaussiana al 90 % y se reparten a partes iguales un 10 % entre mutación de tipo *Grow* y *Shrink.* Esto permite realizar explotación de soluciones, cuya fórmula se parece mucho a la original, aunque esta puede modificarse. Anteriormente, la selección del tipo de mutación a realizar se realizaba utilizando una función de probabilidad con una distribución normal. Además, se incluyó una aproximación de *simmulated annealing* para el operador de mutación de tipo gaussiano, de modo que este tipo de mutación fuese reduciendo el conjunto de valores posibles para un terminal dado en función de su proximidad a su valor. Así este valor sería reemplazado al principio por valores distantes, y se continuaría seleccionando valores más próximos a medida que avance las rondas del proceso evolutivo.

Se utilizan como función a aproximar los valores muestreados de un ciclo de las funciones de cadera eje X (el eje de avance) y rodilla. A diferencia de las ejecuciones de los experimentos anteriores, los resultados se encuentran disponibles en, apenas, cinco minutos, frente a las tres semanas que podían durar, de media, los experimentos anteriores. La mejora de tiempo se debe a que ya no es necesario ni compilar cada fuente, ni arrancar el simulador Webots para cada individuo. Este cambio en la duración de los experimentos permite la probar la idoneidad de los parámetros de configuración, ya que los resultados se encuentran disponibles inmediatamente. La realización de diversas ejecuciones llevó asociado una visualización de las funciones que mejor aproximaban estos conjuntos de valores. Las diferentes ejecuciones mostraron que el error cometido por cada uno de los mejores individuos de las sucesivas poblaciones iba reduciéndose paulatinamente hasta llegar a un mínimo que no se rebasaba.

La visualización de la evolución de las mejores soluciones a lo largo del experimento permitió concluir que se podría estar utilizando un parámetro de profundidad de árbol de-

masiado bajo. Se realizó un aumento de este parámetro al doble de su valor (50) y se vio un mayor ajuste de la mejor función generada a los datos, pero no se pudo comprobar un ajuste mucho mejor hasta que se elevó dicho parámetro a 80. Una ejecución posterior del algoritmo de programación genética utilizando como valores de referencia para la función de *fitness* los correspondientes a la función de la rodilla, confirmó que la configuración también ajustaba bien esta función y se decidió trasladar estos cambios de configuración (la profundidad del árbol de expresión y las probabilidades de aplicación de los diferentes tipos de mutación) y desarrollo (la posibilidad de aplicar una sola mutación de las seis disponibles y distintas en función de una probabilidad) a la aplicación de programación genética original.

Una vez se aplicaron los cambios sobre la aplicación de programación genética general, se ejecuta la simulación con cuatro partes de evolución en dos fases. En la primera fase se prescinde del uso de sensores, mientras que en la segunda éstos sí se incluyen. Al volver a utilizar el dominio de origen, los tiempos de ejecución de los experimentos volvieron a aumentar a los valores anteriores (en el mejor caso, 4 días). Sin embargo, esta vez ya se había realizado una corrección de los parámetros del proceso evolutivo, por lo que los resultados obtenidos cambiaron de forma importante.

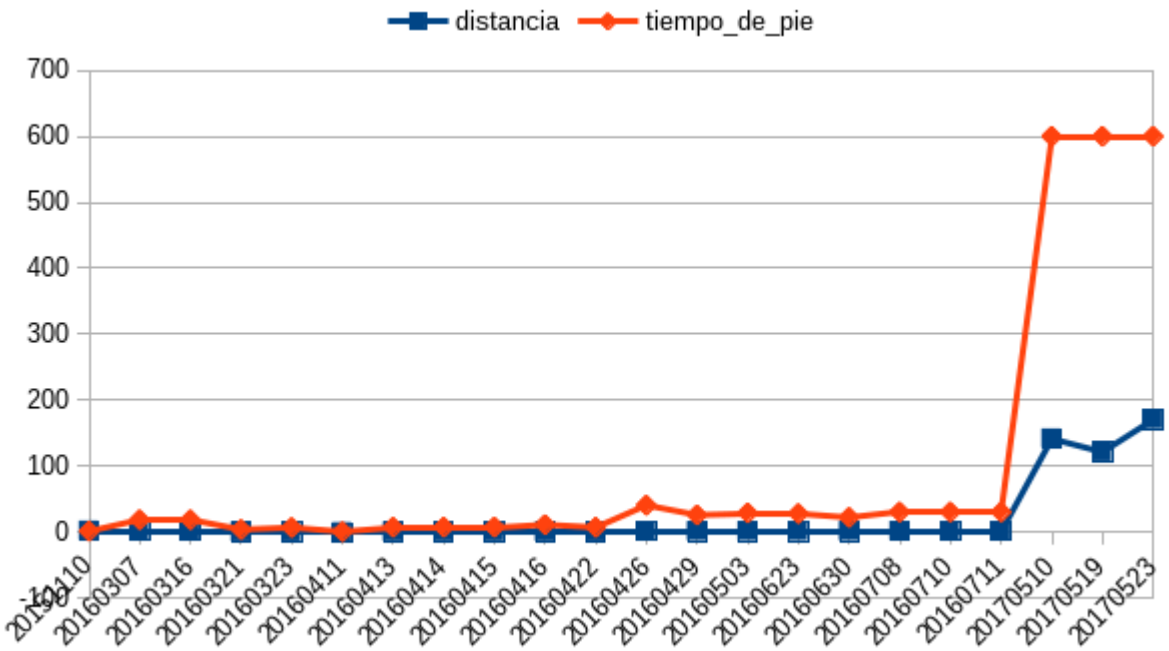


Figura 4.19: Experimentos realizados con el modelo EDNL

Si se presta atención a los resultados obtenidos (ver Figura 4.19), comparados con los obtenidos con el mismo modelo, se observa que los controladores para peatón obtenidos una vez se ha incrementado la profundidad del árbol de expresión hasta 80 nodos de profundidad son más estables y sí producen un movimiento de caminar continuo. Conviene destacar que los resultados se limitaron a 600s, puesto que se tuvo que parar la simulación, pese a que el controlador de peatón utilizado mantenía al peatón erguido y en movimiento en línea recta. Además, se observa que la distancia recorrida para el mejor individuo de 20170523, que utiliza sensores, es ligeramente superior a la obtenida por el mejor individuo de 20170510 ($170,6m$ por $141,45m$), que no los usa. Por tanto, se puede deducir que el uso de sensores le ha permitido al primero andar con estabilidad, a pesar de llevar más velocidad.

4.4. Desarrollo de productos auxiliares

Con objeto de facilitar el desarrollo de este proyecto, se llevaron a cabo diferentes desarrollos adicionales que pasan a enumerarse brevemente a continuación.

4.4.1. Controlador de parámetros de evolución

Los parámetros de una evolución son un factor determinante durante toda la ejecución de un algoritmo evolutivo. En muchos casos, se realiza una cuidada selección de éstos con objeto de mantenerlos a lo largo de todo el experimento.

Sin embargo, también puede optarse por ajustar los parámetros a las condiciones de inicio. En un primer momento se requiere que el algoritmo evolutivo realice una exploración lo más amplia posible del espacio de búsqueda. Para este fin se requiere un porcentaje de mutación elevado ($10\,\%$ - $30\,\%$) y que la presión selectiva no sea muy fuerte (p.e. en el caso de utilizar torneos, emplear un tamaño de torneo de 2).

Una vez se llega hacia la mitad del experimento, las necesidades del mismo difieren de las originales, puesto que en este punto se necesita realizar la explotación de las soluciones encontradas, con objeto de hallar el máximo global. Conviene precisar que la mitad del experimento no se trata de un punto exacto, si no más bien, de una franja de rondas. No obstante, una vez alcanzado este punto teórico, los parámetros del experimento deben

alterarse para permitir la explotación [pasar a un porcentaje de mutación bajo ($1\,\%$ - $5\,\%$) y a una presión selectiva alta (entre 4 y 5, incluso 10 individuos/torneo)].

Además, pueden variarse los parámetros del proceso evolutivo para permitir que éste salga de un posible estancamiento en las cercanías de un máximo local. Si los parámetros no pudiesen cambiarse, no habría posibilidad de influir en el experimento y, por tanto, de conseguir que éste salga del estancamiento, salvo por evolución propia.

Con estas premisas se diseñó una aplicación capaz de procesar la salida del programa evolutivo y mediante reglas, pudiese determinar qué parámetro debía modificar y en qué sentido. En las Figuras 4.20 (abajo-izquierda; abajo-derecha) pueden verse dos pantallazos de la aplicación en ejecución continua en dos rondas distintas (ronda 80 y 93) para determinar el estado de la evolución. En la Figura 4.20 (arriba) puede verse a qué rondas corresponde cada pantalla de la aplicación en una gráfica.

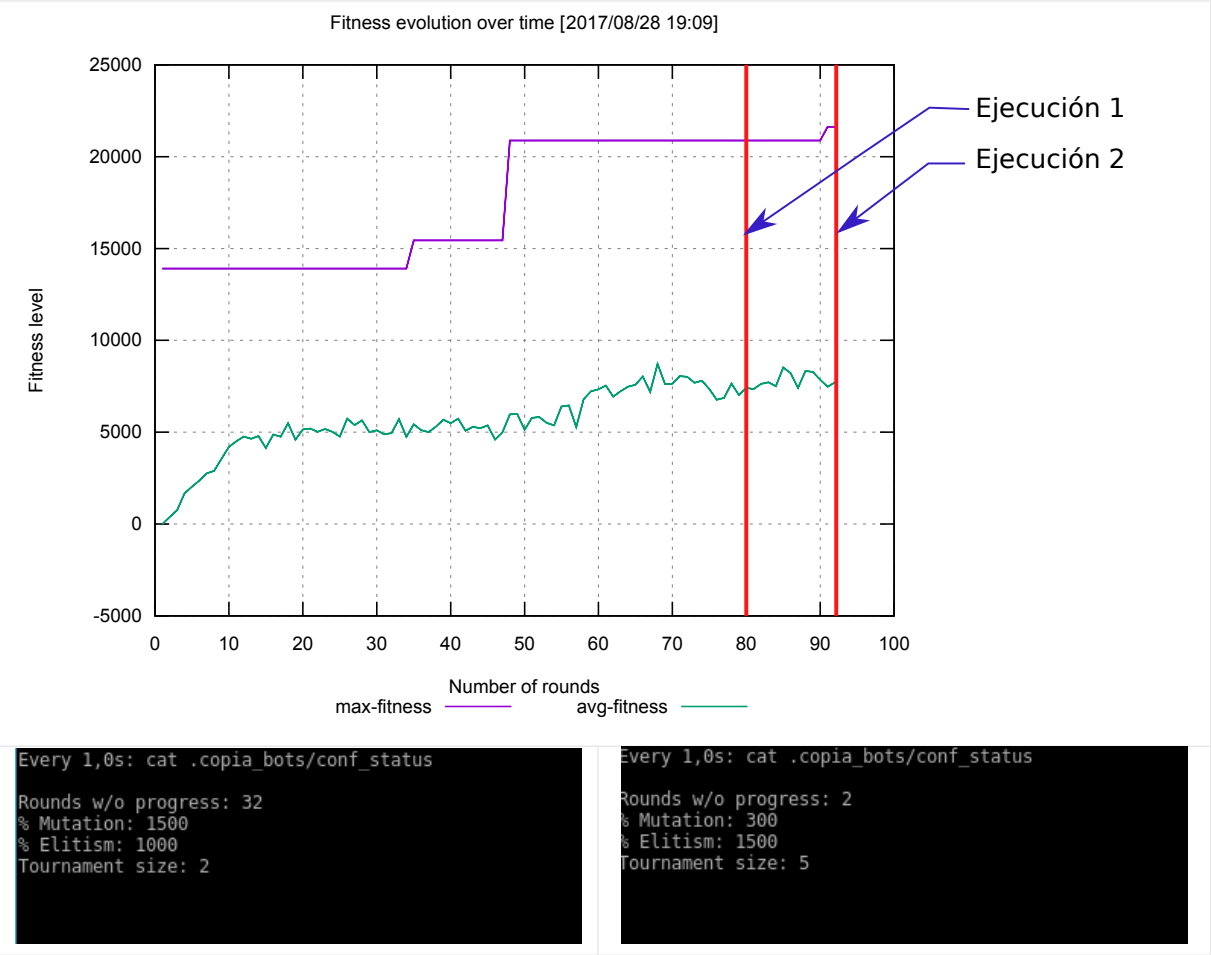


Figura 4.20: Pantallas ilustrativas de la ejecución de la aplicación de control de parámetros. Arriba puede verse marcados en rojo las rondas de las que procede cada pantallazo. En la pantalla abajo izquierda, puede verse el resultado de la ejecución 1. En la pantalla abajo derecha, el resultado de la ejecución 2.

Por ejemplo, si un proceso evolutivo se encontraba estancado durante más de 5 rondas seguidas, el programa de control optaba por aumentar la mutación desde un $1,30\,\%$ a un $5,00\,\%$. Si, posteriormente, se hallaba una solución con mejor *fitness*, éste devolvía el parámetro al $1,30\,\%$ original.

Además, dicha aplicación permitía guardar una copia de aquellos modelos, cuyo *fitness* fuese el más alto encontrado hasta el momento, para poder visualizar las diferentes etapas que ha tenido el proceso evolutivo.

4.4.2. Controlador de duplicados

Determinar si la población dispone de suficiente variabilidad genética es una tarea que puede resultar bastante compleja de realizar con poblaciones pequeñas sin ayuda automática en ejemplos para estudio, pero que se vuelve completamente inviable en cuanto las poblaciones tienen un tamaño suficiente para trabajar con algoritmos evolutivos.

La importancia de conocer que grado de diversidad genética existe en una población radica en que se necesita mantener un alto porcentaje de diversidad como condición necesaria para que el proceso evolutivo no se estanque. Los individuos de una población pueden ser muy parecidos entre sí y disponer de *fitness* similares, pero no debe haber un número elevado de éstos que sean todos copias del mismo. Este hecho podría conducir a un estancamiento prematuro de la población, y que no alcanzase los objetivos propuestos al inicio del experimento.

Para atender esta necesidad, se desarrolló una aplicación de consola capaz de determinar si los fenotipos generados eran distintos y cuántos existían del mismo tipo.

4.4.3. Visor de evolución de ecuaciones diferenciales

Crear una población inicial adecuada al problema se reveló como una tarea de muy difícil consecución. La dificultad se encontraba principalmente en desarrollar un número pequeño de individuos que fuesen capaces de dar algunos pasos (i.e., tuviesen un *fitness* muy alto) e importarlo como un conjunto de superindivíduos para la población inicial. Diseñar a mano el movimiento de las diferentes articulaciones de un individuo capaz de caminar a partir de ecuaciones diferenciales es una tarea costosa. Es difícil desvelar que está causando el desequilibrio que impide al peatón caminar a partir de la observación del movimiento producido por las ecuaciones diferenciales, aunque éste se pueda visualizar a cámara lenta.

Con este problema en mente, se diseñó una solución basada en tuberías y un *framework* de visualización de datos (*matplotlib*, véase [25]). La combinación de ambos permite mostrar la trazada que van generando las ecuaciones diferenciales. En la Figura 4.21 se puede ver una captura del sistema puesto en práctica.

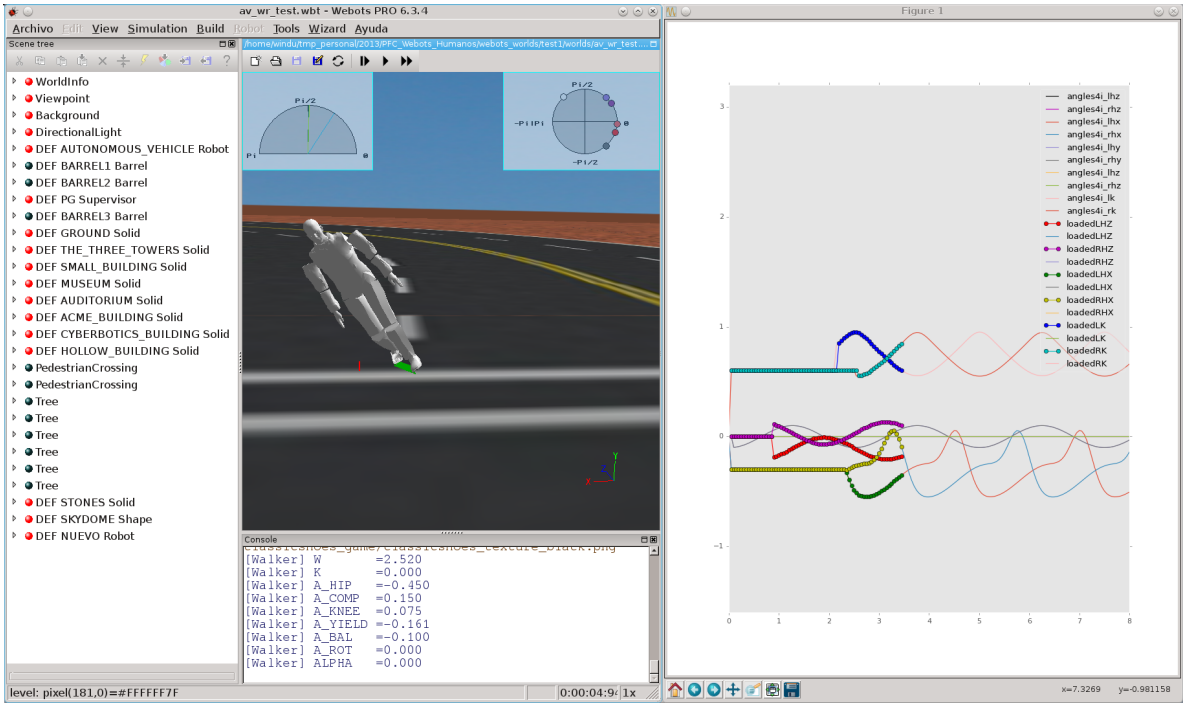


Figura 4.21: Visualizador de ecuaciones diferenciales (der) junto a Webots

La aplicación de este sistema permite predecir el comportamiento de cada articulación controlada vía ecuaciones diferenciales (EEDD), de forma que puede determinarse con mayor precisión la causa del desequilibrio producido en el peatón y puede subsanarse con mayor facilidad. Además, no se necesita ejecutar Webots de forma continua para comprobar cada cambio realizado sobre las ecuaciones diferenciales, puesto que pueden verse los efectos de un cambio haciendo un diagrama de éstas.

El modo de operar con el programa consta de 3 partes diferenciadas y sencillas. En primer lugar, se modifican las ecuaciones diferenciales. En segundo lugar, se modifican los valores de partida de estas. En tercer y último lugar, se comprueban los resultados atendiendo a la ventana de visualización de datos que se abre al ejecutar el programa.

4.5. Resultados

En conclusión, la aplicación del modelo *ad-hoc1* produjo comportamientos que permitían la locomoción bípeda hacia adelante (como muestra su mejor individuo; ver Figura 4.22,

individuo 20140920), aunque ésta carecía de estabilidad y estaba muy limitada en el tiempo.

Los experimentos realizados con el modelo *ad-hoc2* muestran una continuidad con respecto al modelo anterior (individuo 20150110). Este mejor individuo no ofrece una mejora significativa respecto a la distancia recorrida del peatón, ni respecto al tiempo que éste se puede mantener caminando.

El mejor resultado de aplicar el modelo de ecuaciones diferenciales sin correcciones (de los tres tipos aplicados) muestra que el cambio de modelo no produjo necesariamente el controlador deseado, sin una adecuada configuración.

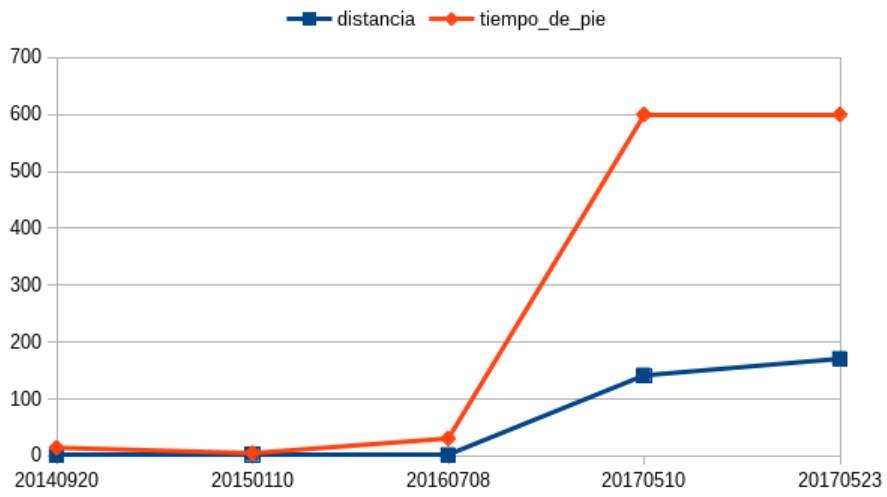


Figura 4.22: Análisis comparativo de mejores individuos obtenidos

Si se presta atención a los mejores resultados obtenidos mediante el modelo de ecuaciones diferenciales con correcciones, se observa que los dos controladores (20170510 y 20170523) para peatón obtenidos son más estables y sí producen un movimiento de caminar continuo. Con estos controladores se consigue el objetivo perseguido de este proyecto. No obstante, se aprecia una diferencia notable entre la distancia recorrida por el controlador que no usa sensores (20170510: $141,5m$) con respecto al controlador que sí los trata (20170523: $170,1m$), que dada la escala de la gráfica puede quedar poco visible.

4.6. Conclusiones

A continuación, se presentan las conclusiones extraídas del trabajo realizado en este capítulo:

- El espacio de búsqueda de soluciones que comprenden todas las técnicas que permiten a un peatón caminar es demasiado complejo para poder efectuar una búsqueda informada vía programación genética, sin facilitar al algoritmo un buen punto de partida.
- Incluso disponer de un buen punto de partida puede no ser suficiente, si no se conocen los valores adecuados o aproximados para los parámetros de configuración de un algoritmo de programación genética.

Capítulo 5

Conclusiones

A partir del trabajo realizado procedo a formular las conclusiones a las que he llegado:

- La implementación de vehículos autónomos requiere de unas pruebas exigentes y no está exenta de fallos, y los errores de estos son difíciles de prever.
- Desarrollar software de simulación que incluya peatones que estén sujetos al motor de física de la aplicación supone a priori un coste elevado, por lo que los peatones se incluyen en este tipo de software como objetos móviles detectables que pueden ser desplazados a lo largo de la simulación. Ahora bien, no se les aplica física ninguna. Se puede detectar el contacto con ellos.
- Implementar una aplicación que utilice programación genética lleva aparejado una inversión de tiempo importante.
- Emplear un modelo bípedo de locomoción reduce el espacio de búsqueda necesario para conseguir locomoción estable, aunque no garantiza el éxito. En muchas de las ejecuciones del algoritmo de PG se empleó el modelo (descrito en [55]) y no se obtuvieron resultados positivos, i.e., que el peatón modelado caminase.
- Los resultados en parte de la experimentación (la relevante al modelo bípedo de locomoción) se han ajustado a lo esperado. La evolución del modelo bípedo basado en ecuaciones diferenciales no lineales ha producido un controlador capaz de hacer caminar un peatón virtual casi en línea recta. Además, el modelo adhoc permitía generar controladores que hacían caminar al peatón, mediante suficiente evolución, pero no de forma continua.

Capítulo 6

Trabajos futuros

Las líneas de investigación futuras que se podrían plantear a raíz de este proyecto pasan por completar el modelo de peatón de forma que:

- Pueda buscar pasos de cebra y cruzar por ellos.
- Pueda pasar a correr mientras se encuentra andando.
- Pueda seguir una trayectoria previamente indicada.
- Pueda cambiar de velocidad caminando, de forma que un vehículo tenga que recalcular la trayectoria, o incluso, detenerse.
- Pueda hacer cambios de rumbo imprevisibles para un conductor, de modo que fuerce al vehículo a reconsiderar el conjunto de posibles acciones a realizar.
- Pueda tropezar llegado un punto, como forma de generar un escenario de peligro alternativo.

Además, se podría:

- Reducir el tamaño del peatón para simular un niño. Esta capacidad permitiría probar los sensores con seres humanos más imprevisibles, de menor tamaño y que suponen un mayor riesgo de fallo para los sensores.

Bibliografía

[1] https://www.youtube.com/watch?v=Mj_S4yLpq1M (verificada a fecha de 2016/06/23) 6, 7, 8, 9, 11, 12

[2] Keirsey, D.; Mitchell, J.; Bullock, B.; Nussmeir, T.; Tseng, D.Y., Autonomous Vehicle Control Using AI Techniques, in Software Engineering, IEEE Transactions on , vol.SE-11, no.9, pp.986-992, Sept. 1985. 14

[3] Maurer, M., Behringer, R., Fürst, S., Thomanek, F., & Dickmanns, E. D. (1996, August). A compact vision system for road vehicle guidance. In Pattern Recognition, 1996., Proceedings of the 13th International Conference on (Vol. 3, pp. 313-317). IEEE. 14

[4] Peck, S., Fatehi, L., Douma, F., & Lari, A. (2015). SDVs Are Coming-An Examination of Minnesota Laws in Preparation for Self-Driving Vehicles, The. Minn. JL Sci. & Tech., 16, 843. 15

[5] https://www.google.com/selfdrivingcar/ (verificada a fecha de 2016/06/23) 17

[6] https://www.youtube.com/watch?v=6LDKf_PLKOo (verificada a fecha de 2016/06/23) 15

[7] Schafer, R. C. (1987). Clinical biomechanics: musculoskeletal actions and reactions. Williams & Wilkins.(Cap. 4). Véase http://www.begin2dig.com/2011/03/different-speeds-have-different.html (verificada a fecha de 2016/06/23) 36

[8] Locomotion. (2011). Encyclopædia Britannica. Encyclopædia Britannica Deluxe Edition. Chicago: Encyclopædia Britannica. 22, 35

[9] http://www.physion.net (verificada a fecha de 2016/06/23). 24

[10] http://www.scuzzstuff.org/oe_cake/ (verificada a fecha de 2016/06/23). 24, 25

[11] http://www.blender.org (verificada a fecha de 2016/06/23). 25, 26

[12] https://msdn.microsoft.com/en-us/library/bb648760.aspx (verificada a fecha de 2016/06/23). 27

[13] https://www.cyberbotics.com/ (verificada a fecha de 2016/06/23). 28

[14] http://www.algoryx.se/products/dynamics-for-spaceclaim/ (verificada a fecha de 2016/06/23). 29, 30

[15] https://www.carsim.com/products/carsim (verificada a fecha de 2016/06/23). 31

[16] https://www.tassinternational.com/prescan (verificada a fecha de 2016/06/23). 31

[17] http://www.oktal.fr/en/automotive/range-of-simulators/professional-solutions (verificada a fecha de 2016/06/23). 32, 33

[18] http://www.vi-grade.com/index.php?pagid=vehicle_dynamics_carrealtime (verificada a fecha de 2016/06/23). 33, 34

[19] Alvarez-Alvarez, A., Trivino, G., & Cordón, O. (2012). Human gait modeling using a genetic fuzzy finite state machine. Fuzzy Systems, IEEE Transactions on, 20(2), 205-223. 38

[20] Koza, J. R. (1992). Genetic programming: on the programming of computers by means of natural selection (Vol. 1). MIT press. 40

[21] Fowler, M., Highsmith, J. (2001) The Agile Manifesto http://agilemanifesto.org/iso/es/ http://www.drdobbs.com/open-source/the-agile-manifesto/184414755(verificada a fecha de 2016/06/23). 50

[22] Succi, G., Marchesi, M., Williams, L., & Wells, J. D. (2002). Extreme programming perspectives. Addison-Wesley Longman Publishing Co., Inc.. 52

[23] Ellnestam, O., & Brolund, D. (2014). The Mikado method (First ed.). Saintmpford;LaVergne;: Manning Publications Company. http://www.methodsandtools.com/archive/mikado.php (verificada a fecha de 2016/06/23). 55

[24] Matos, V., & Santos, C. (2012, November). Central pattern generators with phase regulation for the control of humanoid locomotion. In Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on (pp. 134-139). IEEE. 75

[25] http://matplotlib.org/ (verificada a fecha de 2016/06/23). 92

[26] Clauser, C. E., McConville, J. T., & Young, J. W. (1969). Weight, volume, and center of mass of segments of the human body. ANTIOCH COLL YELLOW SPRINGS OH. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=AD0710622 (verificada a fecha de 2016/06/23). 66, 67

[27] https://www.theguardian.com/technology/2016/mar/09/google-self-driving-car-crash-video-accident-bus(verificada a fecha de 2016/06/23). 17, 18

[28] Bergholz, R., Timm, K., & Weisser, H. (2000). U.S. Patent No. 6,151,539. Washington, DC: U.S. Patent and Trademark Office. 18

[29] http://www.stisimdrive.com/ (verificada a fecha de 2016/06/23). 35

[30] https://www.youtube.com/watch?v=A66zgJ4Oj8o (verificada a fecha de 2016/06/23)

[31] Royce, Winston (1970). "Managing the Development of Large Software Systems", Proceedings of IEEE WESCON, 26 (August): 1-9. 51

[32] ISO/IEC 14882:2003. International Organization for Standardization. 51

[33] ISO/IEC 14882:2011. International Organization for Standardization. 51

[34] End User License Agreement, Microsoft 52

[35] Navarro Corrales, Mariano (2007). “Práctica de computación biológica nº3: Programación genética y Robocode” 2, 59

[36] http://www.makehuman.org/ 68

[37] https://www.blender.org/ 69

[38] http://www.meshlab.net/ 71

[39] Especificación de VRML versión 1.0 70, 71

[40] https://www.youtube.com/watch?v=tbEUziUPtZ0 35

[41] https://www.youtube.com/watch?v=E5Rll_Q-zEs 32

[42] Geddes, N. B. (1940). Magic Motorways. https://ia801405.us.archive.org/2/items/magicmotorways00ge a fecha de 2017/08/11). 14

[43] Wallace, R. (1985). First results in robot road-following. JCAI'85 Proceedings of the 9th international joint conference on Artificial intelligence. 14

[44] Dempster, W. T., & Gaughran, G. R. (1967). Properties of body segments based on size and weight. Developmental Dynamics, 120(1), 33-54. 66, 67

[45] Drillis, R., Contini, R., & Bluestein, M. (1966). Body segment parameters. Research Division, NY: New York University, School of Engineering and Science. 66, 67

[46] https://www.ald.softbankrobotics.com/en/robots/nao 65

[47] https://pr.fujitsu.com/en/news/2001/09/10.html 65

[48] https://es.wikipedia.org/wiki/Malla_poligonal 68

[49] https://en.wikipedia.org/wiki/Robotics_simulator

[50] http://www.sae.org/misc/pdfs/automated_driving.pdf 13, 101

[51] http://newatlas.com/sae-autonomous-levels-definition-self-driving/49947/ 101

[52] Composición del autor a partir de traducción propia de [50, 51] 14

[53] https://www.blender.org/foundation/history/ 26

[54] Silva, P., Santos, C. P., Matos, V., & Costa, L. (2014). Automatic generation of biped locomotion controllers using genetic programming. Robotics and Autonomous Systems, 62(10), 1531-1548. 39, 87

[55] Matos, V., & Santos, C. (2012, November). Central pattern generators with phase regulation for the control of humanoid locomotion. In Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on (pp. 134-139). IEEE.

39, 82, 96

Apéndice A

Listado de software puesto a disposición pública

El software desarrollado que se va a dejar disponible para descarga comprende las siguientes aplicaciones:

- Aplicación de programación genética desarrollada.
- Aplicación de visualización de datos como salida de Webots. (Live)
- Aplicación de control de ejecución.

Apéndice B

Manual de instalación del software

A continuación, se indica como se debe instalar el software para poder trabajar con él. En primer lugar, se debe disponer de una instalación LINUX para poder trabajar con la aplicación. Además, se deben comprobar los siguientes parámetros:

- Se dispone de gcc versión entre 4.0 y 5.0.
- Se dispone de la librería boost-program_options versión 1.55.
- Se dispone de la librería xml2 con versión 2.9.1.
- Se dispone de Webots en la versión 6.3.4. Con otras versiones, no se ha podido confirmar que funcionase.

En segundo lugar, debe crearse un controlador Walker a través de Webots y anotar la ruta para más adelante. También debe crearse un controlador supervisor, con nombre RobotEvaluator, para poder calcular el *fitness* de cada individuo.

En tercer lugar, se debe crear un fichero de configuración gp.cfg de acuerdo a lo dispuesto en el anexo C, en el mismo directorio donde se encuentre el ejecutable de RobotGenerator.

En ultimo lugar, debe ajustarse el parámetro *render-dir* en el fichero de configuración gp.cfg recién creado para indicar la ruta al controlador Walker.

Apéndice C

Manual de usuario

A continuación se presenta el manual de usuario.

C.1. Cómo ejecutar el programa

La aplicación desarrollada es una aplicación de consola para entorno Linux. Para ejecutar la aplicación se debe ejecutar el siguiente comando:

Consola linux

```
[user@localhost ~] $ ./RobotGenerator
```

Figura C.1: Modo de ejecución sin población inicial predefinida

O si se desea utilizar una población de partida no aleatoria:

Consola linux

```
[user@localhost ~] $ ./RobotGenerator -c
```

Figura C.2: Modo de ejecución con población inicial ya predefinida

En ultima instancia, se presenta el fichero de configuración y se muestran en él las opciones configurables.

Fichero de configuración gp.cfg

```
# Specify how long the experiment should run.
# Choose a number of rounds.
# This parameter may be changed at runtime
iterate-for = 100

# Specify the crossover rate.
# Choose a value between 0 and 10000 (100,00)
# This parameter may be changed at runtime
crossover-rate = 8000

# Specify the population mutation rate
# Choose a value between 0 and 10000 (100,00)
# This parameter may be changed at runtime
pop-mutation-rate = 130

# Specify the gen mutation rate
# Choose a value between 0 and 10000 (100,00)
gen-mutation-rate = 3000

# Specify the elitism rate
# Choose a value between 0 and 10000 (100,00)
# This parameter may be changed at runtime
elitism-rate = 1500

# Specify the tournament size
# Choose a positive value
# This parameter may be changed at runtime
tournament-size = 10

# Specify the population size
# Choose a positive value
population-size = 500

# Specify the number for the starting round
# Choose a positive value
starting-round = 1
```

Fichero de configuración gp.cfg (cont)

```
# Specify the number for the first robot
# Choose a non-negative value.
first-robot = 0

# Specify the location for the CSV files for the initial population
# Any valid path will suffice
# Only available if continue-exp has been selected and model type is
# articulation
robot-dir = ./_robot_maestro

# Specify the location for the XML file for the initial population
# Any valid path to a valid XML file will suffice
# Only available if continue-exp has been selected and model type is
# function or diffeq
robot-file = ./robots.20160322.xml

# Specify the location where the controller should be rendered
# Any valid path to a Webots controller directory is accepted
render-dir = ../../webots_worlds/test1/controllers/Walker/

# Specify the rate that a tree would be shrinked or enlarged when
# applying tree mutation
# Choose a value between 0 and 10000 (100,00)
height-change-rate = 3000

# Specify the number of mutation points
# Choose a positive number or zero
mutation-num-points = 1

# Specify the number of crossover points
# Choose a positive number or zero
crossover-num-points = 1

# Specify the number of mutation points
# Choose a positive number or zero
mutation-type = tree
```

Fichero de configuración gp.cfg (cont)

```
generator-type = function
robot-type = haploid
gen-min-limit-size = 600
gen-max-limit-size = 5800
gen-size-matters = false
steady-state = false
loop-min-time = 9500
loop-max-time = 10500
model-type = diffeq
crossover-type = pure

# Specify the number of parts the chromosome will have
# Choose a positive number
num-parts = 12

# Specify the number of articulation
# Choose a positive number
num-articulations = 1

# Specify the rate that a terminal would be chosen over a non terminal
# symbol
# Choose a value between 0 and 10000 (100,00)
terminal-over-nonterminal = 5000
```

C.2. Cómo trabajar con la salida

La salida de la aplicación presenta el siguiente formato de forma estándar (Figura C.3).

```
Population size: [500]
Steady-state mode: No
Elite number of individuals:50
Tournament size: 10
Should continue prev experiment: Yes
Crossover rate       : [8000]
Population mutation rate: [3000]
Gen mutation rate: [3000]
Render directory: ../../webots_worlds/test1/controllers/Walker/
Model type: [diffeq]
Number for first created robot: [0]
Starting round: [1]
Robots file: [./robots.20160503.xml]
Iterate for: [1] rounds up to [100]
Number for first created robot: [0]
Height change rate: [3000]
Number of points in mutation: [1]
Number of points in crossover: [1]
Robot type: [haploid]
Crossover type: [pure]
Slow-start: [1]
Generator type: [function]
Mutation type: [function]
Numero de alelos minimo por gen por defecto: [1]
Numero de alelos maximo por gen por defecto: [25]
Number of parts the source code will have: [4]
Number of articulations simulated: [1]
Choose T over NT: [5000]
Generator type:rampedhalfhalf
RobotGenerator was last compiled on 2016-04-29 at 14:11:35
Running RobotGenerator on 2016-05-03 at 21:26:08
First created robot: [0]
Valid robots: [23]
Robots not created yet: [477]
Joining...
Evaluating...
[bot0] Fitness: 554 / 1000000000 [0.00%] [ numErrors: 0]
[bot1] Fitness: -2796 / 1000000000 [-0.00%] [ numErrors: 0]
[bot2] Fitness: 6070 / 1000000000 [0.00%] [ numErrors: 0]
[bot3] Fitness: 6467 / 1000000000 [0.00%] [ numErrors: 0]
```

Figura C.3: Captura de la salida del inicio de la aplicación

Sin embargo, dicha salida no resulta muy amistosa para el usuario, puesto que cuesta seguir la evolución de la aplicación a lo largo del tiempo. Para producir una salida más fácil de comprender, se ha realizado un tratamiento de salida compuesto de 3 fases. En primer lugar, se calcula la media de los *fitness* de todos los individuos generados por cada ronda. Después, se ha buscado el mejor *fitness* para cada ronda y el código de identificación de ese individuo. A continuación, se ha generado un fichero compuesto con este formato (tabla C.1). Por último, se emplea este fichero como entrada para la aplicación *gnuplot* que permite generar una salida más fácilmente interpretable (figura C.4).

Tabla C.1: Formato fichero para entrada de *gnuplot*

ronda	*fitness* máximo	*fitness* medio	código mejor individuo

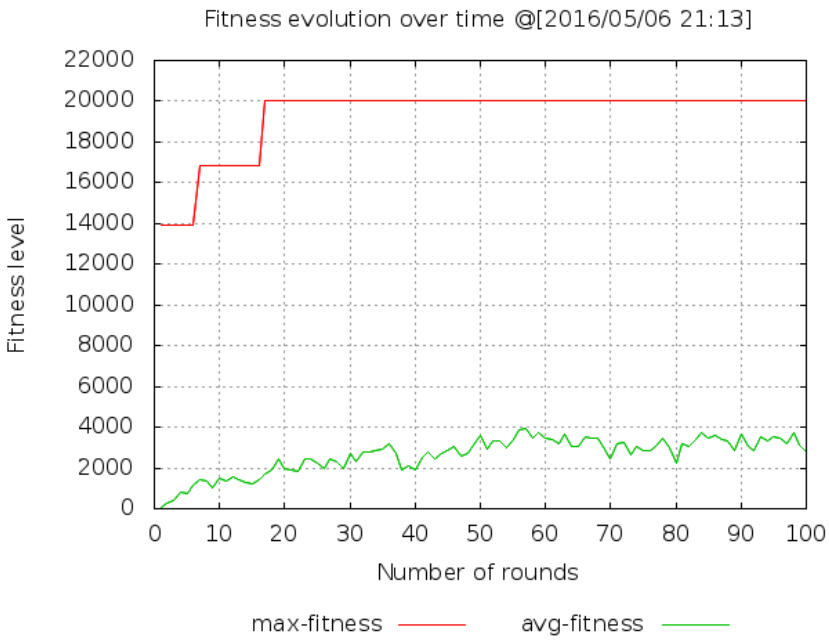


Figura C.4: Captura de la salida mejorada de la aplicación

Apéndice D

Listado de experimentos realizados

Tabla D.1: Listado de experimentos ejecutados en 2017.

	Fecha exp	TT	% C	%M	%E	TP	NGG	MFGM	MFGE	VEL (m/s)	
1	20170523	2	0,00%	100,00%	10,00%	1000	43	6195	306320	170,63	600
2	20170519	2	0,00%	100,00%	10,00%	1000	19	6195	151775	121,47	600
3	20170510	2	0,00%	100,00%	10,00%	1000	49	6195	502253	141,45	600

Tabla D.2: Listado de experimentos ejecutados en 2016.

	Fecha exp	TT	% C	%M	%E	TP	NGG	MFGM	MFGE	VEL (m/s)	
1	20160711	2	0,00%	60,00%	15,00%	500	150	13908	21625	1,51	30,5
2	20160710	10	80,00%	90,00%	15,00%	500	33	13908	13908	1,51	30,5
3	20160708	10	80,00%	1,30%	15,00%	500	58	13908	13908	1,48	30,5
4	20160630	1	80,00%	30,00%	15,00%	500	100	13908	16029	1	22,4
5	20160623	10	80,00%	1,30%	15,00%	500	100	13908	18777	1,04	28,1
6	20160503	10	80,00%	30,00%	10,00%	500	100	13908	20064	1,02	29,1
7	20160429	10	80,00%	3,00%	10,00%	500	124	6832	18555	1,03	26
8	20160426	10	80,00%	30,00%	10,00%	500	87	7655	18012	1,47	40,8
9	20160422	10	80,00%	30,00%	10,00%	500	100	4708	24354	0,09	7,3
10	20160416	10	0,00%	30,00%	10,00%	500	100	4708	41662	0,24	11,4
11	20160415	10	80,00%	30,00%	10,00%	500	33	4708	4708	0,22	7,7
12	20160414	10	80,00%	30,00%	10,00%	500	24	4708	4708	0,2	7,6
13	20160413	2	80,00%	30,00%	10,00%	500	29	4708	4708	0,19	7,5
14	20160411	2	80,00%	30,00%	10,00%	500	18	0	0	-0,6	0,8
15	20160323	2	80,00%	30,00%	10,00%	500	100	3641	4356	0,13	7
16	20160321	2	80,00%	30,00%	10,00%	500	38	-420	828	0,13	4
17	20160316	2	80,00%	30,00%	10,00%	500	100	43726	45152	1,91	18,8
18	20160307	10	80,00%	1,30%	15,00%	500	100	48493	48493	1,92	18,8

Tabla D.3: Listado de experimentos ejecutados en 2015.

	Fecha exp	TT	% C	%M	%E	TP	NGG	MFGM	MFGE	VEL (m/s)	
1	20151110	5	85,00%	3,00%	15,00%	1000	81	1483	30270	1,86	1,5
2	20150325	5	65,00%	3,00%	15,00%	100	177	0	822342307	-0,96	0,1
3	20150324	5	65,00%	3,00%	15,00%	100	176	0	243151350	1,4	0,4
4	20150323	5	65,00%	5,00%	15,00%	100	416	0	17148128800	0,67	1,3
5	20150322	5	65,00%	5,00%	15,00%	100	110	0	290702500	0,96	0,2
6	20150317	5	65,00%	10,00%	15,00%	100	137	0	1649984400	-0,65	0,4
7	20150207	dañad	dañado	dañado	dañado	dañado	4000	0	3,2E+15		
8	20150126	5	80,00%	30,00%	15,00%	100	3435	0	6,66875E+15	1,55	3
9	20150119	5	100,00%	1,30%	15,00%	100	1754	0	4,61671E+15	2,16	3,5
10	20150116	no existe									
11	20150110	5	100,00%	20,00%	15,00%	100	1204	1,1E+016	2,98E+016	2,28	4,7
12	20150119	5	100,00%	1,30%	15,00%	100	1754	0	4,61671E+15	2,16	3,5
13	20150126	5	80,00%	30,00%	15,00%	100	3435	0	6,66875E+15	1,55	3

Tabla D.4: Listado de experimentos ejecutados en 2014.

	Fecha exp	TT	% C	%M	%E	TP	NGG	MFGM	MFGE	VEL (m/s)	
1	20141219	5	100,00%	1,30%	15,00%	100	3684	1,01E+016	2,33E+016	2,16	4,3
2	20141205	5	100,00%	1,30%	15,00%	100	2866	1,01E+016	2,88E+016	2,32	4,7
3	20141125	5	100,00%	1,30%	15,00%	100	2527	1,15E+016	2,50E+016	2,12	4,5
4	20141118	5	100,00%	1,30%	15,00%	100	825	2,28E+015	6,30E+015	1,52	2,8
5	20141112	5	100,00%	1,30%	15,00%	100	1298	1,15E+016	1,47E+016	1,87	4
6	20141107	-	100,00%	10,00%	15,00%	100	1076	5,96E+015	1,16E+016	1,76	3,8
7	20141022	-	100,00%	1,30%	15,00%	100	4000	5,96E+15	1,23E+016	2,19	4
8	20141005	-	N/A	N/A	15,00%	100	2705	2,75E+015	1,15E+016	1,51	3,3
9	20141004	-	N/A	N/A	15,00%	100	157	1,28E+12	2,28E+015	1,69	3
10	20140921	-	N/A	N/A	15,00%	100	301	3,49E+014	3,49E+014	2,66	4,2
11	20140920	-	N/A	N/A	15,00%	100	301	0	1,48E+014	1,25	14,2
12	20140919	-	N/A	N/A	15,00%	100	301	0,00E+000	3,11E+014	2,29	4
13	20140918	-	N/A	N/A	15,00%	100	301	494900	1,31E+007	3,72	5,4
14	20140916	-	N/A	N/A	15,00%	100	301	4,95E+005	7,63E+006	3,51	5
15	20140915	-	N/A	N/A	15,00%	100	301	494900	6,25E+007	6,38	5
16	20140913	-	N/A	N/A	15,00%	100	301	4,95E+005	1,32E+007	0,93	2,1

Apéndice E

Diseño de componentes

E.1. Aplicación de programación genética

E.1.1. Tipo

Este componente es un módulo.

E.1.2. Función

La función de este módulo es llevar a cabo la evolución de las diferentes poblaciones de individuos hasta que se alcance el criterio de *fitness* seleccionado o una ronda objetivo.

E.1.3. Interfaces

Las interfaces que utiliza este componente se listan a continuación:

- IEvaluate:
    - Entrada: Código del individuo a evaluar y nombre del controlador que se utiliza en la evaluación.

    - Salida: *fitness* del individuo y penalizaciones asociadas.

- IConfig:

    - Entrada: Parámetros facilitados por la linea de comandos y nombre del fichero de configuración.
    - Salida: Valor adecuado para cada uno de los parámetros.

E.1.4. Dependencias

Este componente depende a su vez de los siguientes componentes:

- CID#2: Le permite evaluar los individuos de cada población.
- CID#3: Le da acceso a los parámetros de configuración.

E.1.5. Procesamiento

El procesamiento de este componente se describe de forma abstracta en la Figura E.1.

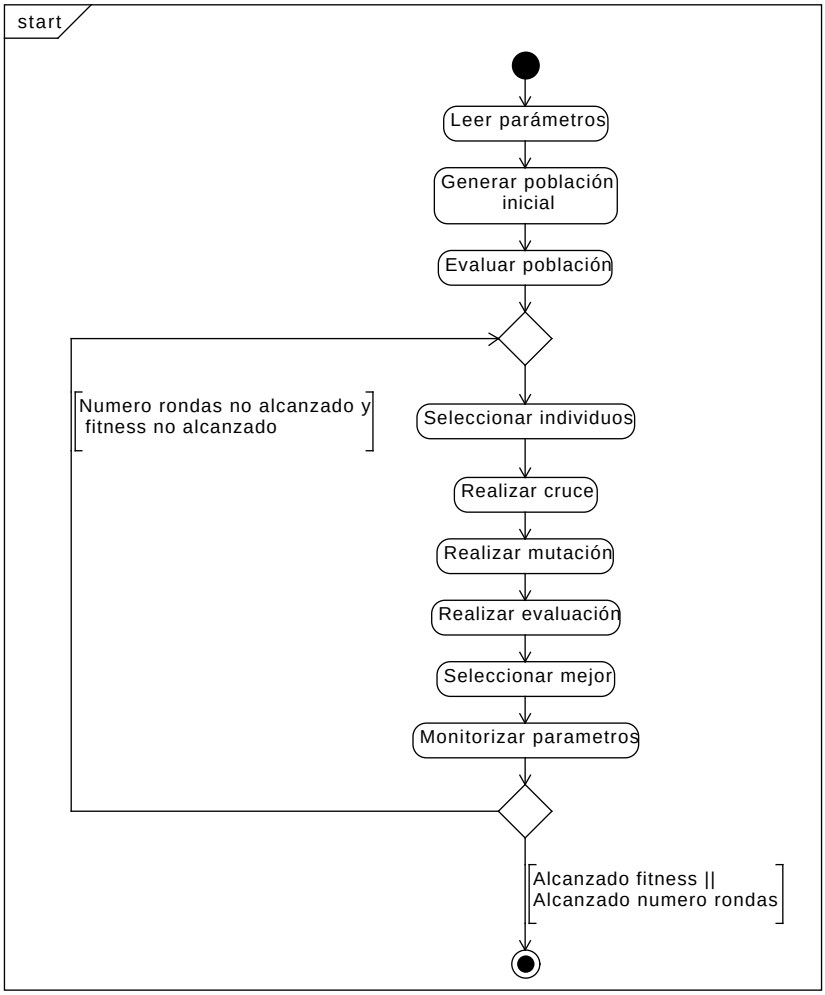


Figura E.1: Procesamiento del componente CID#1

E.1.6. Datos

E.1.7. Recursos

E.1.8. Estructura de paquetes

E.2. CID#1 - Aplicación de evaluación webots

E.2.1. Tipo

Este componente es un módulo, que se apoya en una aplicación externa para ofrecer la funcionalidad de evaluación.

E.2.2. Función

E.2.3. Interfaces

E.2.4. Dependencias

E.2.5. Procesamiento

E.2.6. Datos

E.2.7. Recursos

E.3. CID#1 - Liberia de acceso a la configuración

E.3.1. Tipo

Este componente es un módulo, que se apoya en una librería externa para ofrecer la funcionalidad de gestión de los parámetros facilitados a la aplicación.

E.3.2. Función

E.3.3. Interfaces

E.3.4. Dependencias

E.3.5. Procesamiento

E.3.6. Datos

E.3.7. Recursos

Apéndice F

Interfaces empleadas

F.1. IEvaluator

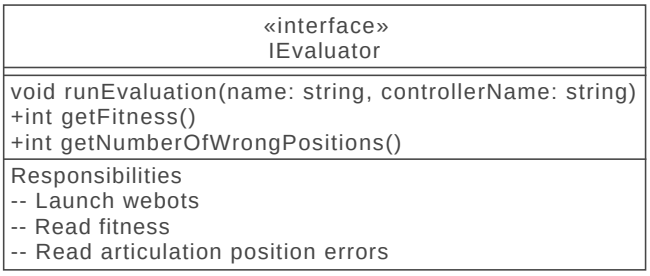


Figura F.1: Interfaz IEvaluator

La responsabilidad de la interfaz IEvaluator consiste en evaluar individuos y recuperar el *fitness* del individuo, así como recuperar también el número de posiciones erróneas que han tenido las articulaciones simuladas. Para llevar a cabo esto, la interfaz ejecuta de forma síncrona la aplicación externa Webots.

A continuación, se explican los diferentes métodos de la interfaz IEvaluator (Figura F.1) .

runEvaluation

Esta función realiza la evaluación del individuo. Recibe como parámetro el identificador del individuo y el nombre del controlador de *Webots* que se va a utilizar. El primer parámetro también se utiliza para dar nombre a los ficheros generados a partir del individuo.

En primer lugar, traslada el fuente generado al directorio destino del controlador. En segundo lugar, realiza su compilación. En ultimo lugar, lanza la ejecución de *Webots* de forma síncrona.

Debido a la versión de *Webots* disponible, solamente se puede lanzar una instancia de Webots a la vez. Esto se debe a que no es posible indicarle a *Webots* desde la linea de comandos ni el directorio de trabajo, ni que simulación debe correr. Por lo que el proceso de evaluación se ve forzado a ejecutarse de forma secuencial.

getFitness

Esta función recupera el *fitness* de la evaluación del fichero de resultados especificado en el controlador.

getNumberOfWrongPositions

Esta función recupera el *fitness* de la evaluación del fichero de resultados especificado en el controlador.

F.2. IConfig

Figura F.2: Interfaz IConfig

La responsabilidad de la interfaz IConfig radica en recibir los parámetros a través de fichero de configuración y linea de comandos y formar una configuración valida para ejecución. Además, deja disponible los diferentes parámetros de configuración a través de métodos get con un nombre similar.

Por último, también es su cometido controlar las modificaciones del fichero de configuración para permitir modificar parámetros de configuración en tiempo de ejecución.

getInstance Permite obtener la instancia que contiene la configuración.

clear Permite restaurar la configuración a valores por defecto.

parseOptions Permite obtener la configuración tratando la linea de comandos, así como el fichero de configuración definido.

getGenerator Permite obtener el valor del parámetro

getCrossoverRate Permite obtener el porcentaje de cruce.

getPopMutationRate Permite obtener el porcentaje de mutación de la población.

getGenMutationRate Permite obtener el porcentaje de mutación de cada individuo.

getIterateFor Permite obtener el número de rondas que se debe iterar.

getElitismRate Permite obtener el porcentaje de elitismo.

getEliteNumInd Permite obtener el número de individuos que deben copiarse de una población a otra en concepto de elitismo. Se trata de un parámetro derivado.

isConfigured

isSteadyState Permite obtener el modo de ejecución del algoritmo de programación genética.

getTournamentSize Permite obtener el tamaño del torneo en la fase de selección.

getLastRound Permite obtener cual es la ultima ronda que debe ejecutarse. Se trata de un parámetro derivado del número de rondas a ejecutar.

getRobotFile Permite obtener el nombre del fichero que contiene los robots de partida en formato XML. Este formato solo sirve para obtener robots cuyo movimiento de articulaciones este dirigido por una fórmula matemática.

getRobotDir Permite obtener el directorio donde se encuentran los robots de partida en formato CSV. En cada fichero CSV sólo habrá un robot, en el que cada bloque de articulaciones se corresponde con una fase o step en el movimiento del robot. Este modo de importación solo es compatible si se utiliza el tipo de operación de articulación.

getRenderDir Permite obtener el directorio donde se deben compilar los robots. Corresponde con el directorio del controlador del robot humanoide en Webots.

getFirstRobotToCreate Permite saber en que valor se empiezan a contar los robots que se generen con la aplicación de programación genética.

getPopulationSize Permite saber de que tamaño deben ser las poblaciones durante la evolución.

getGeneratorType Permite saber que tipo de generador debe utilizarse en la generación inicial.

getMutationType Permite saber el tipo de mutación que debe aplicarse.

getHeightChangeRate Permite saber que porcentaje de cambio se debe aplicar si se utiliza un tipo de mutación basado en árbol.

getMutationNumPoints Permite saber cuántos puntos de mutación deben aplicarse en una mutación de un individuo.

getCrossoverNumPoints Permite saber cuantos puntos de cruce deben aplicarse para cruzar dos individuos.

getRobotType Permite saber si deben utilizarse robots haploides o diploides. Por defecto, se utilizaran robots haploides (con un solo cromosoma). En el caso de los diploides, cada robot contendrá dos cromosomas.

getWeakGenRate Solamente se aplica si se utilizan robots diploides. Permite saber en qué porcentaje debe utilizar el gen débil en lugar del gen fuerte.

getGenMinLimitSize Permite saber el tamaño mínimo que debe tener el gen.

getGenMaxLimitSize Permite saber el tamaño máximo que debe tener el gen.

getLoopMinTime Solamente se aplica en el caso de que se utilice el modelo de operación articulación. Este parámetro permite saber que duración mínima debe tener el bucle de operación del robot.

getLoopMaxTime Solamente se aplica en el caso de que se utilice el modelo de operación articulación. Este parámetro permite saber que duración máxima debe tener el bucle de operación del robot.

getModelType Permite saber el tipo de modelo que se desea utilizar. Se puede seleccionar entre los modos:

- Operación. Utiliza cada operación para indicar la articulación y la apertura de esta.
- Step. Similar al anterior, pero utiliza cada step como si fuese una parte de una secuencia del cromosoma.
- Función. Utiliza la operación para definir una ecuación. El movimiento viene definido por los valores producidos por la ecuación.
- Ecuación diferencial. Similar al modo anterior, pero en este caso la función descrita es la derivada de la función objetivo. El movimiento de la articulación viene determinado por la posición anterior y un incremento calculado en función de la derivada.

getCrossoverType Permite saber el tipo de cruce que se desea aplicar.

getNumArticulations Permite saber el número de articulaciones que se van a evolucionar.

getNumParts Permite saber el número de partes en las que se encontrará dividido el cromosoma.

shouldContinueExp Permite saber si se debe importar una población inicial o se genera una población de cero.

getChooseTerminalOverNonTerminal Permite saber la ratio que debe aplicarse al escoger entre terminal o no terminal durante el proceso de generación de un nuevo árbol de operación.

isSlowStart Permite saber si se va a utilizar el modelo de slow-start, en el que las ecuaciones diferenciales se inicializan en 0 y linealmente se va aumentando los valores fruto de la ecuación diferencial, hasta terminar el periodo slow start.

Apéndice G

Diseño de paquetes

En esta subsección se procede a describir los diferentes componentes de los que se compone la aplicación desarrollada de forma jerárquica. Dicha subsección dispone de un mayor nivel de detalle técnico, puesto que el nivel de detalle de las descripciones facilitadas es a bajo nivel.

Con objeto de facilitar la descripción y la comprensión, se facilita a continuación un identificador a cada componente:

Tabla G.1: Relación de identificador/componente del sistemas

Código	Componente referenciado
CID#01	Master
CID#02	Phases
CID#02.1	Generator
CID#02.2	Evaluator
CID#02.3	Selection
CID#02.4	Crossover
CID#02.5	Mutation
CID#03	Util
CID#03.1	SrcCodeGenerator
CID#03.2	Importer
CID#03.3	Config

G.1. CID#01 - Master

El componente master se encarga de coordinar las diferentes fases del proceso de programación genética, así como de realizar una salida coordinada de los resultados que se obtienen a lo largo del proceso.

G.1.1. Tipo

Este componente es un módulo.

G.1.2. Función

Tal y como se ha descrito anteriormente, realiza tareas de coordinación de las fases del proceso para dirigir el proceso evolutivo.

G.1.3. Interfaces

G.1.4. Dependencias

Este componente depende a su vez de los siguientes componentes:

- CID#02 - Phases: Este componente le facilita acceso a las diferentes fases que puede utilizar en el proceso evolutivo, así como a sus diferentes variantes.
- CID#03 - Util: Este componente le permite importar individuos previamente elaborados, exportarlos para su compilación, así como tratar la linea de comandos de forma adecuada para leer opciones de uso.

G.1.5. Procesamiento

En lineas generales, el procesamiento realizado en este componente corresponde con la gestión de las poblaciones existentes (origen y destino), así como invocar a las diferentes fases del algoritmo de programación genética.

En pseudocódigo, el proceso podría describirse del siguiente modo:

```
Generar individuos de partida
Evaluar individuos
Hasta cumplir A rondas o alcanzar fitness propuesto
    Llamar a operador de selección B veces
    Llamar a operador de cruce C veces
    Llamar a operador de mutación D veces
    Evaluar los B individuos generados
Mostrar fitness mejor individuo
```

G.1.6. Datos

Con objeto de desarrollar las tareas descritas, el componente requiere acceso a los datos descritos en el diagrama XXX. En éste diagrama se muestra que este componente dispone de un operador para cada fase del algoritmo de programación genética, así como los individuos generados de la población anterior y la actual.

G.1.7. Recursos

Los recursos necesarios para realizar el trabajo de este componente se limitan a tener acceso a la configuración de la aplicación y la linea de comandos.

G.2. CID#02 - Phases

El componente phases aglutina las diferentes fases que requiere un algoritmo de programación genética y las diferentes implementaciones para cada uno de ellas.

G.2.1. Tipo

Este componente es un módulo.

G.2.2. Función

La función del componente phases consiste en proporcionar acceso a las diferentes fases que tiene un algoritmo de programación genética. Además, también ofrece acceso a las diferentes técnicas que pueden aplicarse para realizar una misma fase.

G.2.3. Interfaces

Este componente utiliza las siguientes interfaces:

- Entrada: Recibe uno o varios árboles de expresión y una operación a aplicarles.
- Salida: Mismo número de árboles de expresión, una vez aplicado sobre ellos la operación indicada.

G.2.4. Dependencias

Este componente depende a su vez de los siguientes componentes:

- CID#03 - Util: Este componente le permite obtener los parámetros necesarios para configurar las diferentes fases de un algoritmo de programación genética, ya sean para elegir la técnica a emplear en cada fase, o bien, para especificar los parámetros de configuración de ésta.

G.2.5. Procesamiento

G.2.6. Datos

G.2.7. Recursos

G.3. CID#02.1 - Generator

Este subcomponente es una parte integrante del componente CID#02, que aglutina las fases del algoritmo de programación genética. Este componente, en particular, agrupa las diferentes modalidades de generadores que se han empleado en este proyecto.

G.3.1. Tipo

Este componente es un módulo.

G.3.2. Función

Este componente lo forman en exclusiva generadores de árboles de expresión. Su cometido radica en generar de forma aleatoria árboles que puedan evolucionarse posteriormente. Entre ellos se encuentran las modalidades:

- Full.
- Grow.
- Ramped half & half.

G.3.3. Interfaces

Este componente utiliza las siguientes interfaces:

- Entrada: Recibe la profundidad del árbol a desarrollar.
- Salida: Genera un árbol con la profundidad. Atendiendo a la técnica escogida, puede ser un árbol completo o no.

G.3.4. Dependencias

Este componente depende a su vez de los siguientes componentes:

- CID#03.3 - Config: Este componente le permite obtener los parámetros necesarios para configurar la fase de generación de un algoritmo de programación genética, ya sean para elegir la técnica a emplear, o bien, para especificar los parámetros de configuración de ésta.

G.3.5. Procesamiento

Dependiendo de la técnica escogida, el procesamiento de este componente puede variar ligeramente.

G.3.6. Datos

Los datos que emplea este componente constan de:

- Los parámetros que controlan el proceder de la técnica.
- Los árboles generados como resultado de la ejecución.

G.3.7. Recursos

En principio, este componente no requiere de recursos externos.

G.4. CID#02.2 - Evaluator

G.4.1. Tipo

G.4.2. Función

G.4.3. Interfaces

G.4.4. Dependencias

G.4.5. Procesamiento

G.4.6. Datos

G.4.7. Recursos

G.5. CID#02.3 - Selection

Este subcomponente es una parte integrante del componente CID#02, que aglutina las fases del algoritmo de programación genética. Este componente, en particular, agrupa las diferentes modalidades de selección que se han empleado en este proyecto.

G.5.1. Tipo

Este componente es una clase.

G.5.2. Función

El objetivo de este componente consiste en seleccionar un individuo de una población de forma aleatoria y estocástica. Permite que el algoritmo de programación genética seleccione los individuos de forma aleatoria, que formaran las siguientes poblaciones.

G.5.3. Interfaces

Este componente utiliza las siguientes interfaces:

- Entrada: Recibe el número de individuos que integran el torneo.
- Salida: Devuelve una copia del individuo que tiene mejor *fitness* de entre los seleccionados para el torneo.

G.5.4. Dependencias

Este componente depende a su vez de los siguientes componentes:

- CID#03.3 - Config: Este componente le permite obtener los parámetros necesarios para configurar la fase de selección, para especificar el tamaño de torneo que se quiere emplear.

G.5.5. Procesamiento

El proceso que sigue este componente a grandes rasgos se especifica a continuación.

```
Mientras no se llegue al máximo número de individuos
    Seleccionar uno aleatoriamente
Seleccionar individuo con fitness mas alto
```

A pesar de que este componente implementa la técnica de torneos, existen otras técnicas que podrían emplearse, tal como la ruleta, que no se han utilizado en este proyecto para realizar la fase de selección de individuos.

G.5.6. Datos

Los datos que emplea este componente constan de:

- El parámetro que controla el tamaño del torneo.
- El conjunto de individuos existentes.

G.5.7. Recursos

Este componente no requiere de recursos externos para operar.

G.6. CID#02.4 - Crossover

Este subcomponente es una parte integrante del componente CID#02, que aglutina las fases del algoritmo de programación genética. Este componente, en particular, agrupa las diferentes modalidades de cruce que se han empleado en este proyecto.

G.6.1. Tipo

Este componente es un módulo.

G.6.2. Función

El objeto de este componente consiste en aplicar un intercambio de subárboles entre los distintos árboles de entrada.

G.6.3. Interfaces

Este componente utiliza las siguientes interfaces:

- Entrada: Recibe dos árboles de expresión para cruzar y la probabilidad de que ambos se crucen.
- Salida: Devuelve dos árboles que se incluyen en la nueva población a generar.

G.6.4. Dependencias

Este componente depende a su vez de los siguientes componentes:

- CID#03.3 - Config: Este componente le permite obtener los parámetros necesarios para configurar la fase de cruce, es decir, la técnica a emplear para realizar el cruce, así como los parámetros que la configuran, tales como el porcentaje de individuos que se cruzan cada ronda.

G.6.5. Procesamiento

El procesamiento de este componente puede describirse de forma general tal que:

```
Se listan los nodos disponibles en cada uno de los dos árboles.
Se escoge un nodo para cruzar de cada lista.
Se intercambian los nodos escogidos.
Se devuelven los árboles modificados.
```

G.6.6. Datos

Los datos que emplea este componente constan de:

- El parámetro que controla el porcentaje de individuos que se cruzan.
- La pareja de individuos que se van a cruzar.

G.6.7. Recursos

Este componente no requiere de recursos externos para operar.

G.7. CID#02.5 - Mutation

Este subcomponente es una parte integrante del componente CID#02, que aglutina las fases del algoritmo de programación genética. Este componente, en particular, agrupa las diferentes modalidades de mutación que se han empleado en este proyecto.

G.7.1. Tipo

Este componente es un módulo.

G.7.2. Función

El objeto de este componente es modificar de forma parcial o total un árbol de expresión, de modo que el resultado que produzca sea distinto.

G.7.3. Interfaces

Este componente utiliza las siguientes interfaces:

- Entrada: Recibe un árbol de expresión para mutar y la probabilidad de que éste se mute, así como el porcentaje de mutación que debe sufrir.
- Salida: Devuelve un árbol nuevo incluido en la población nueva.

G.7.4. Dependencias

Este componente depende a su vez de los siguientes componentes:

- CID#03.3 - Config: Este componente le permite obtener los parámetros necesarios para configurar la fase de mutación, es decir, la técnica a emplear para realizar la mutación, así como los parámetros que la configuran, tales como el porcentaje de individuos que se mutan cada ronda.

G.7.5. Procesamiento

El procesamiento de este componente puede describirse de forma general tal que:

```
Se listan los nodos disponibles del árbol.
Se escoge un nodo para mutar de la lista.
Se elimina el nodo escogido.
Se genera un nodo nuevo de tamaño similar al borrado.
Se devuelve el árbol modificado.
```

G.7.6. Datos

Los datos que emplea este componente constan de:

- El parámetro que controla el porcentaje de individuos que se mutan.
- El individuo que se va a mutar.

G.7.7. Recursos

Este componente no requiere de recursos externos para operar.

G.8. CID#03 - Util

Este componente es uno de los componentes principales y agrupa diversas funciones de apoyo al algoritmo de programación genética.

G.8.1. Tipo

Este componente es un módulo.

G.8.2. Función

Las funciones de las que se encarga este componente son 3:

1. Facilita acceso a la configuración del experimento.
2. Permite importar robots para la población de partida.
3. Permite generar los ficheros fuentes de los individuos que representan.

G.8.3. Interfaces

G.8.4. Dependencias

G.8.5. Procesamiento

G.8.6. Datos

G.8.7. Recursos

G.9. CID#03.1 - SrcCodeGenerator

El componente *SrcCodeGenerator* es un componente miembro del paquete CID#03 - Util. Este componente en particular agrupa las diferentes formas de traducir el conjunto de fórmulas de un individuo en un programa ejecutable.

G.9.1. Tipo

Este componente es un módulo.

G.9.2. Función

La función de este componente es crear un ejecutable (o fenotipo) a partir de un individuo (o genotipo).

G.9.3. Interfaces

Este componente utiliza las siguientes interfaces:

- Entrada: Recibe un individuo (un conjunto de árboles de expresión).
- Salida: Produce un fichero con un programa compilable.

G.9.4. Dependencias

Este componente depende a su vez de los siguientes componentes:

- CID#03.3 - Config: Este componente le permite obtener los parámetros necesarios para configurar la plantilla para el fichero fuente a compilar. Además, permite indicar que plantilla fuente se desea utilizar.

G.9.5. Procesamiento

En lineas generales, el procesamiento que se realiza en este componente se podría describir con el siguiente procedimiento en pseudocódigo:

```
Añadir 1º parte del fichero
Traducir formula 1
...
Traducir formula n
Añadir 2º parte del fichero
```

G.9.6. Datos

El componente accede a los siguientes datos para poder realizar sus funciones:

- Conjunto de literales para formar cada plantilla de fichero fuente.
- Conjunto de formulas en representación arbórea de las que dispone un individuo.

G.9.7. Recursos

Este componente requiere acceso de escritura al disco para poder escribir los ficheros fuentes que se van a compilar.

G.10. CID#03.2 - Importer

El componente *Importer* es un componente miembro del paquete CID#03 - Util. Este componente en particular agrupa las clases que permiten importar un robot, ya sea a partir de un fichero XML o un fichero CSV.

G.10.1. Tipo

Este componente es un módulo.

G.10.2. Función

El componente tiene por objeto importar individuos a partir de ficheros XML o CSV. En función del formato del individuo, la importación se realiza de un modo u otro.

1. Si se trata de un individuo formado por un árbol de articulaciones, con sus respectivos ángulos, éste debe importarse utilizando ficheros CSV.

2. Si, por el contrario, se trata de un individuo formado por formulas matemáticas, éste debe importarse utilizando ficheros XML. En este caso, las formulas deben escribirse en sintaxis JAVA, y el compilador matemático generará el correspondiente árbol a la fórmula indicada.

Ambos sistemas son incompatibles entre sí, puesto que los diferentes tipos de individuo se corresponden con modos diferentes de ejecución de la aplicación.

G.10.3. Interfaces

Este componente utiliza las siguientes interfaces:

- Entrada: Recibe un conjunto de ficheros CSV o un sólo fichero XML.
- Salida: Produce un conjunto de individuos correctos.

G.10.4. Dependencias

Este componente depende de:

- Librería libxml2, para procesamiento de ficheros XML en C++.
- Programas bisonc++ y flexc++, como generadores del compilador de expresiones matemáticas.

G.10.5. Procesamiento

El procesamiento de este componente se describe a continuación. En primer lugar se describe el funcionamiento en caso de tratarse de un fichero XML:

```
Hasta leer fin de fichero, leer individuo:

    Para cada individuo:
```

```
            Para cada cromosoma:
               Para cada parte:

                  Importar formula y generar árbol correspondiente
```

En caso de tratarse de un fichero CSV, el procesamiento varia para importar dichos individuos:

```
    Mientras no lea fin de fichero:
        Mientras la linea no esté en blanco o fin de fichero:
            Mientras la linea no sea step

                Incorporar la articulación y ángulo al árbol del step.
            Pasar al step siguiente
        Pasar al individuo siguiente
```

G.10.6. Datos

Este componente solamente accede a:

- las opciones de configuración, y
- las descripciones de los individuos, ya sea en forma de fórmula o como listado de articulaciones y ángulos.
- los nodos de los individuos generados.

G.10.7. Recursos

Este componente requiere acceso de lectura a los ficheros CSV, o al fichero XML, donde se encuentre las descripciones de los individuos.

G.11. CID#03.3 - Config

El componente CID#03.3 Config es un componente miembro del componente CID#03 - Util. Este componente analiza y almacena la configuración del experimento durante la ejecución del mismo.

G.11.1. Tipo

Este componente es una clase.

G.11.2. Función

Este componente se encarga de gestionar las opciones de configuración. Permite leer dichas opciones a partir de la linea de comando, o bien a partir de un fichero.

G.11.3. Interfaces

Este componente ofrece la interfaz IConfig, que se describe en detalle en el apéndice F.2.

G.11.4. Dependencias

Este componente depende de:

- librería boost_program_options, que se utiliza para extraer los parámetros de configuración deseados a partir de una combinación de un fichero de configuración y la linea de comandos.

G.11.5. Procesamiento

El procesamiento que se realiza en este componente se puede describir con el siguiente pseudocódigo:

```
Analizar la entrada y el fichero de configuración
Para cada variable definida:

    Obtener su valor
    Configurar la opción asociada con el valor obtenido.

Si ya existe una configuración:


    Comprobar si los valores existentes se han modificado en esta lectura de la configura
```

G.11.6. Datos

Este componente solamente accede a los datos que conforman su clase.

G.11.7. Recursos

Este componente requiere acceso a los parámetros de la linea de comando, así como al fichero de configuración gp.cfg, donde se indica la configuración que se aplicará al experimento.

Apéndice H

Modo de detección de fugas de memoria

Las fugas de memoria son un error cometido que tiene consecuencias de diversa índole en la estabilidad de la aplicación. Estás pueden derivar en cuestiones menos importantes, tales como un consumo de memoria mayor de lo esperado, o en cuestiones de mayor importancia, tales como el fin de la ejecución de la aplicación por un acceso a memoria ilegal.

Con el fin de eliminar los errores en la gestión de la memoria, se ha utilizado la aplicación *valgrind.*

La aplicación se ha ejecutado de tres formas distintas:

- $ valgrind –tool=memcheck ./RobotGenerator

  Este forma de ejecución realiza un análisis simple de la memoria solicitada por el programa a lo largo de una invocación. Permite descartar fallos de memoria de partida.

- $ valgrind –tool=memcheck –leak-check=full ./RobotGenerator

  Esta forma de ejecución realiza un análisis en profundidad. Realiza una comprobación más precisa de la memoria solicitada y no devuelta.

- $ valgrind –tool=memcheck –leak-check=full –track-origins=yes ./RobotGenerator

Esta forma de ejecución realiza un análisis en profundidad al igual que la anterior. La diferencia radica en que este método busca identificar qué llamadas al sistema solicitaron la memoria no devuelta.